博士論文

Assembly of nucleotide sequences of the multipartite structures in plant mitochondria genomes and the diploid human genomes

(植物ミトコンドリアゲノムおよび 2倍体ヒトゲノムの塩基配列のアセンブリ)

舛谷 万象

Assembly of nucleotide sequences of the multipartite structures in plant mitochondria genomes and the diploid human genomes

(植物ミトコンドリアゲノムおよび 2倍体ヒトゲノムの塩基配列のアセンブリ)

Bansho Masutani
舛谷 万象

(Supervisor: Professor Shinichi Morishita)
(指導教員:　森下真一　教授)

A Dissertation

Submitted to
Department of Computational Biology and Medical Sciences
Graduate School of Frontier Sciences
The University of Tokyo

In Partial Fulfillment of the Requirements
for the Degree of Doctor of Science

On December 7, 2022

Abstract

Genome assembly is the process of reconstructing the DNA sequence of a genome from observed subsequences (*reads*). It is one of the fundamental processes in biology because genome sequences are powerful tools to analyze how we have evolved from ancestors, how genomes relate to phenotype, and how much genetic diversity we have.

As a result of advances in technology and algorithms, the gapless assembly of a human haploid genome and nearly complete assemblies of other organisms are now available. These assemblies are called *reference genomes*. We typically analyze genomes of interest (e.g., individual genomes in a population) by representing them as single nucleotide variants (SNVs) and structural variations (SVs) on the references.

However, the difference between a genome sequence of a sample and a reference sequence can be so high that there are no obvious ways to represent the genome as SNVs and SVs on the reference. For example, the mitochondrial genomes in plants recombine considerably from strain to strain and organelle to organelle, shaping multipartite structures. These recombinations are complex, and SVs are insufficient to represent them (Fig. 1a). They are worth investigating because some genes in these genomes inhibit a plant from making mature pollen, called cytoplasmic male sterility. To fully represent the plant mitochondrial genomes, we need to assemble the genome and catalog the recombinations in each strain.

Another example is the major histocompatibility complex (MHC) region and the leukocyte receptor complex (LRC) region in human genomes. These immune-related regions have diverged, and the two haplotypes in an individual often have significant differences, which are hard to represent as SNVs and SVs on a single reference (Fig. 1b). Although many variants in these regions show correlations with diseases, the responsible genes are still ambiguous due to these differences and high linkage disequilibrium. To understand these regions, we need the complete assembly for each haplotype of these regions. In short, the first topic is:

a genome is more than a set of SNVs and SVs on a reference.

I focus on (1): multipartite structures of the mitochondrial genomes in the eight strains of *Arabidopsis thaliana*, and (2): the MHC and the LRC region in two human diploid genomes.

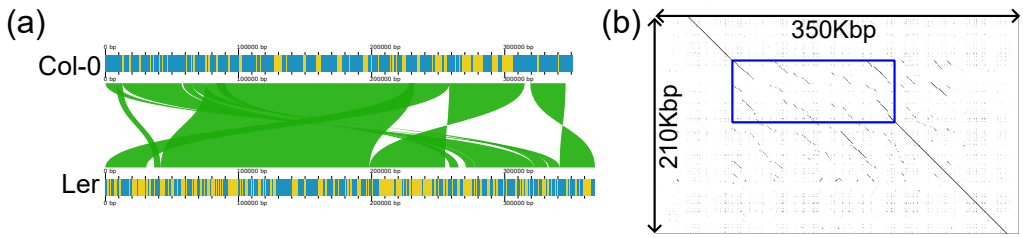

Figure 1: Examples of highly diverged regions in plant mitochondrial genomes (Col-0 and L*er* strain, a) and the human MHC region (two haplotypes, a).

Nevertheless, these cases contain nearly identical regions in addition to highly divergent regions. Specifically, the plant mitochondrial genome recombines frequently, but the mutation rate is low. Similarly, the MHC and LRC regions contain nearly identical regions with as low as 0.1% difference between haplotypes.

As sequencing technologies improve, DNA sequencers produce reads of tens to hundreds of thousands of base pairs, which can span these nearly identical regions. However, high sequencing error rates in these reads (5-15%) blur true SNVs in these regions and make it difficult to distinguish SNVs from the sequencing errors. Can we correctly assemble the highly diverged regions and the nearly identical regions simultaneously by the reads with high error rates? This is the second topic I answer positively by implementing software named JTK. In short,

genome assembly of difficult regions is possible with error-prone long reads.

JTK – regional genome assembler

JTK assembles a target region, such as the MHC region in a diploid genome, from erroneous Oxford Nanopore Technology (ONT) or PacBio reads. It takes three steps to assemble highly diverged regions and nearly identical regions simultaneously.

1. JTK samples 2000-bp subsequences (*chunks*) from the reads. Then, it aligns these chunks to the reads with a relaxed sequence similarity threshold so that multiple copies of nearly identical regions in the target region are represented as the same chunk. JTK captures the SVs and highly diverged regions through these alignments by the presence or absence of the chunks in the reads. Also, JTK estimates the copy number of each chunk in the target region by using how many times it is aligned to the reads.

2. JTK finds SNVs on these chunks and separates the chunks into homologous copies. To exhaustively enumerate SNVs on a chunk, JTK introduces each possible SNV to the chunk and accepts it as an actual SNV if the alignment scores of many reads increase. I developed a sampling-based algorithm to separate these SNVs into each copy. JTK further improves the separation on a chunk by integrating the results of nearby chunks.

3. JTK constructs a graph representing the highly diverged regions found in the first step and the SNVs inferred by the second step and produces the assembly by traversing the graph. JTK is available at GitHub (https://github.com/ban-m/jtk).

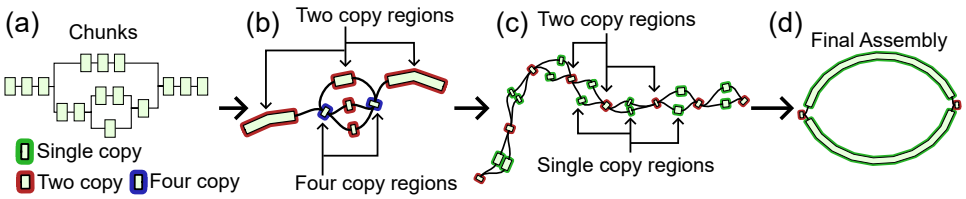

Figure 2: Overview of the JTK assembler

Complex genome arrangements in mitochondrial genomes in the six strains of *Arabidopsis thaliana*.

I assembled the major conformations of the mitochondrial genomes in the nine datasets, consisting of eight strains and one biological replicate, from 350-fold reads with ~15 % errors sequenced by PacBio Sequel sequencer. These assemblies were highly concordant with the reference in the base pair level.

Nonetheless, there were many recombinations (Fig. 3). Their positions were close to the repetitive sequences in the genomes, confirming that the repeat-mediated recombinations have promoted these complex arrangements in this species. Also, I investigated the alignments between the reads and the assemblies and found putative recombinations and linear structure in the multipartite structures. These assemblies and the datasets are available at GitHub (https://github.com/ban-m/reconstruct_mito_genome).

Haplotype differences in the MHC and the LRC regions in human genomes.

JTK produced fully resolved assemblies for the MHC and the LRC region in HG002 and B080 samples (a Japanese B-cell) from 60-fold ONT reads, while other software based on the same datasets produced sub-optimal assemblies. The contiguities and the base-level accuracy of the JTK's assemblies were on par with those produced from high-coverage PacBio HiFi reads, Hi-C reads, and ultra-long ONT reads. The assemblies on the HG002 are available at Zenodo (https://doi.org/10.5281/zenodo.7192214).

In the LRC region in B080, the two haplotypes contained genes of the major haplotype in the Japanese population. However, there were around 0.2% substitutions between the haplotypes on average, and there was a 3Kbp expansion of a

CT-rich region. Also, in the MHC region in the B080 (Fig. 4), a segmental duplication occurred in the class II region. These results revealed considerable variations among the haplotypes in the LRC region, which have been considered the same haplotype in course-grained resolution.

Conclusion

I could assemble the highly diverged regions with error-prone reads. In plant mitochondrial and human diploid genomes, newly assembling the genome provided insights into these genomes. Soon, complete genome sequences will be available on a population scale, and interpretation of them will be a fascinating field of research.

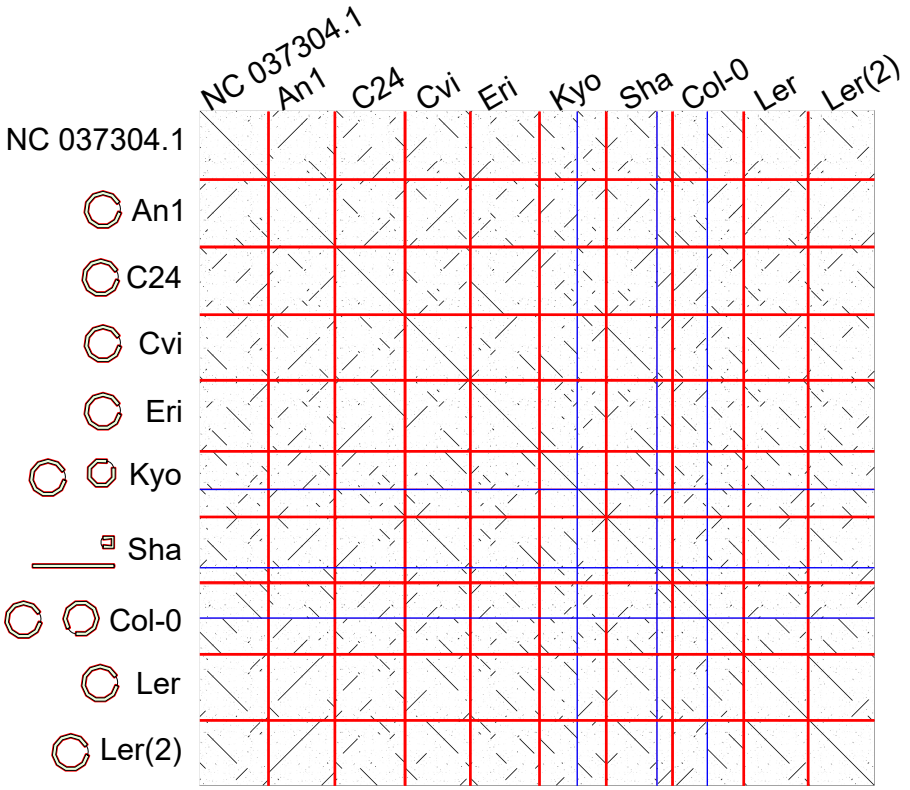

Figure 3: The dotplots between the major structures of mitochondrial genomes in eight strains. The first sequence, NC_037304.1, is the reference sequence, and the rest are assemblies produced by JTK. I assembled two L*er* strain as a biological replicates (L*er* and L*er*(2)). The thick red lines indicate the boundaries between strains, and the thin blue lines indicate the boundaries of assemblies in a strain. On the vertical axis, I put the assembly graph for each strain. For visualization purposes, I added edges connecting the end of the contig to the start of the contig if the contigs were circular.

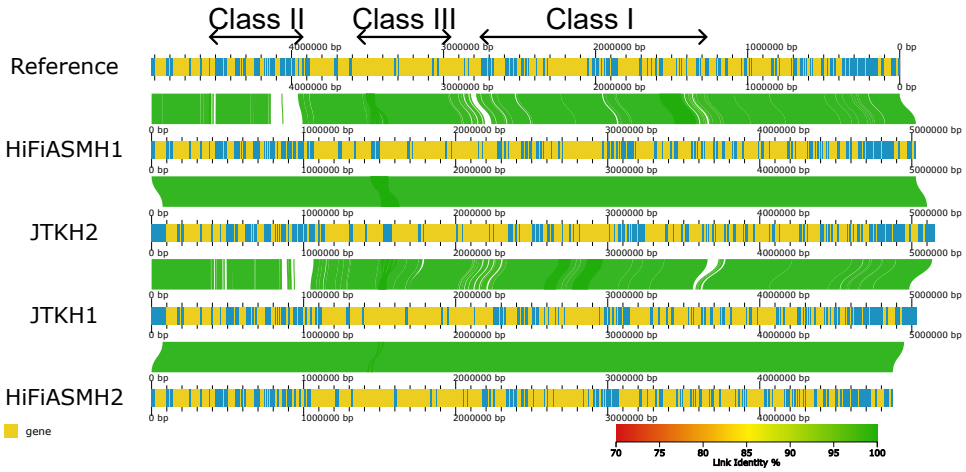

Figure 4: Assemblies on the MHC region in the Japanese sample. I compared the reference sequence, the haplotypes assembled `HiFiASM`, and the haplotypes assembled by `JTK`. The bars consisting of yellow bands (genes) and blue bands (intergenic regions) indicate the assemblies, and the green ribbons between assemblies are alignments.

Acknowledgements

To begin with, I would like to thank my supervisor Prof. Shinichi Morishita. Without his continuous support, warm comments, and invaluable advice, I could not have accomplished my Ph.D. course.

I greatly appreciate everyone who kindly spared time for my doctoral degree, including the reviewers of this thesis: Profs. Masahiro Kasahara, Martin Frith, Tetsuo Shibuya, Shin-ichi Arimura, and Shinichi Morishita.

I also would like to my collaborators, Drs. Shin-ichi Arimura, Yuta Suzuki, and Yoshihiko Suzuki. They endured my somewhat unendurable laziness and gave numerous insightful comments.

I am very thankful to other members of the Morishita lab; Jun Yoshimura, Qu Wei, Chie Owa, Haruka Kobayashi, Ryought Nakabayashi, Rikihumi Ota, Yuki Ichikawa, Riki Kawahara, Hiroki Ito, and Boyuan Son (honorific titles omitted). Even though we had different research topics, the conversation frequently sparked ideas for the issues we tackled.

I would appreciate the support from the WINGS for Sustainable Agriculture Education Program and JSPS. Without their support, I would not even have entered the Ph.D. course. Dear Ministry of Finance and Ministry of Education, Culture, Sports, Science, and Technology, it is essential to ensure finalcial support for students. Please do not cut the budget and keep supporting the future.

Contents

Chapter 1

General introduction

Know thyself

An Ancient Greek aphorism

1.1 Why genomes? Why not?

1.1.1 Abstract

All the genetic information inherited from an individual is called the individual's *genome*. Nowadays, it also refers to the DNA (deoxyribonucleic acid) molecules consisting of four types of bases in the cells. We represent a genome as a string written in four characters corresponding to these bases. The genomic sequence plays a central role in many fields of research due to their simplicity, even though there is additional information, such as the methylation of a DNA molecule. In this thesis, I will argue how to reconstruct genomes on computers.

1.1.2 What is a genome?

A genome includes DNA sequences

What is a genome? The Scitable by Nature Education defines a genome as "the complete set of genetic information in an organism" ([99]). The Oxford learner's dictionary defines it as "the complete set of genes in a cell or living thing" ([108]). The National Human Genome Research Institute gives another definition. It defines a genome as "the entire set of DNA instructions found in a cell" ([52]).

Although these definitions seem intuitive at first glance, they use terms such as "instructions", "genetic information", or "genetic material." These terms are abstract and do not specify any physical entry we can observe. To elucidate the property of the genome through experiments, we need another concrete definition of the "genome."

After the long lines of research, the DNA (deoxyribonucleic acids) molecules in a cell were revealed as genetic materials (see Chapter 1 of ([19])). Briefly, DNA is a pair of nucleic acids, and nucleic acids are polymers of nucleotides consisting of 5-carbon sugars, phosphate groups, and nitrogenous bases. Schematically,

$$\begin{cases} \text{5-carbon sugers} \\ \text{phosphate groups} \\ \text{nitrogenous bases} \end{cases} \longrightarrow \text{nucleotides} \longrightarrow \text{DNA}$$

There are usually four nitrogenous bases, leading to four types of nucleotides. They are called adenine, cytosine, guanine, and thymine. [1] I represent these four bases as `A` (adenine), `C` (cytosine), `G` (guanine), and `T` (thymine) and write down a DNA as a string consisting of these four characters.

I define these DNA molecules in a cell as the "genome" of an organism. As explained later in this section, by adopting this concrete definition, researchers have tried to reveal how our genomes are similar, how genomes have evolved, and how a genome functions in a cell.

Nonetheless, almost everybody dealing with genomes knows that DNA molecules are not everything an organism passes to its offspring, and representing a genome as a DNA string is a merely useful approximation of the genome. What is missing from this approximation, and how harmful is it?

1.1.3 Passing information through non-DNA media

In addition to DNA molecules, living things have other ways of transmitting information from parents to children. [2]

For example, *Bicoid* and *Nanos* are well-known genes playing central roles in oogenesis, i.e., the maturing process of eggs, in *Drosophila melanogaster* ([41, 32]). The proteins encoded by these two genes have different concentrations in different parts of an egg, determining the egg's anterior and posterior direction ([1]). These gradients of the concentration are information to determine the development of an individual, and the information is inherited from parents to children. Thus, it satisfies the condition to be called "genomic."

In addition, organisms modify their DNA to respond to the external environment or to differentiate into different cell types during development, which is called *epigenome* ([123, 80, 126]).

For example, a hydrogen atom (-H) in a nucleotide in a DNA molecule can be replaced by a methyl group (-CH3), which is called *DNA methylation*. These

[1] Almost everything has exceptions in biology. Concerning DNA, researchers have created new types of bases in DNA molecules ([34]).

[2] We have languages to pass information to the future. Insects have their nest to accumulate their work. Bacteria create biofilms for the next generations. They are information from parents to children, but they are not genetic because they are not inherited by reproduction or cell division. Thus, I do not include them in genomes.

methylated DNAs act as marks in the DNA that the transcription and translation mechanisms can use. Notably, part of the epigenome is inherited from parent to child ([69]), making these modifications part of the "genome".

These two examples show that it is not sufficient to consider a string of DNA as the genome itself (Fig. 1.1(a)). A recent paper describes other examples ([112]), and I note that there are still active arguments about the degree and medium of the inheritance of them([120, 49]).

Also, DNA molecules in somatic cells are not inherited by subsequent generations (Fig. 1.1 (a)). So, there is a risk that our findings are actually the non-heritable ones if they are derived from the DNA molecules in somatic cells.

Given these examples, the definition of the genome seems to be a "big tent", and it may not be appropriate to use the word (genome) to refer to DNA sequences, let alone the "Human Genome Project", which apparently did not examine the proteins in fertilized eggs.

However, I advocate the paradigm of considering DNA as the primary source of an individual's heritable information, at least so far. In my opinion, considering the DNA sequence as the genome is a simple but useful *model* that makes many algorithms applicable and has produced fruitful results (Fig. 1.1(b)).

Even though DNA molecules usually have regions that are difficult to determine, and we sometimes determine DNA sequences incorrectly, this representation makes virtually all text processing algorithms applicable to "genomes". Accepting that the representation is missing some regions and contains artifacts, genome researchers and bioinformaticians have improved the model, i.e. the genome sequences, and obtained many fruitful results (Fig. 1.1(c)).

In the following section, I will give some examples of these results.

1.1.4 Why are genomes important?

A lot of research depends on genome sequences, and I can not pick all these researches exhaustively. Instead, I pick four research fields to stress the importance of genomes.

Genome-wide association studies

Some families are more likely to have a specific disease than other families. If this tendency is due to a genomic condition, we call this disease *hereditary disease*. These diseases can strongly correlate with variations in these families' genomes. Some researchers think it is essential for everyone to know the risk of these diseases they have in their genomes and have developed methods to find the relationships between genomes and diseases. One of the most famous – or notorious – examples is a certain kind of breast cancer caused by a mutation in the *BRCA1* or *BRCA2* gene ([90, 155]).

However, finding a variation on a genome that relates to a specific disease is not easy in general because human genomes are too large (~3 billion bases) and

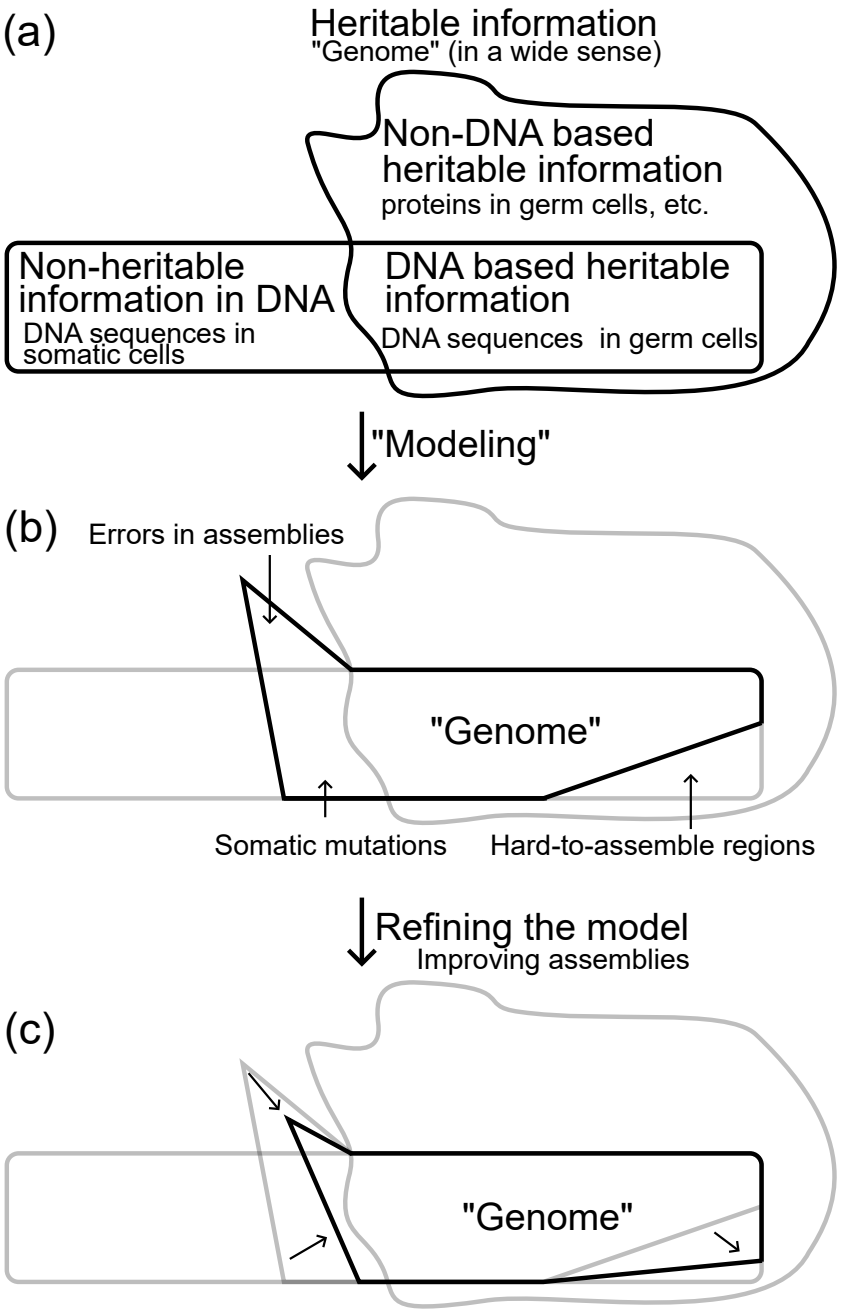

Figure 1.1: Model the genome as DNA sequences.
(a): Heritable information consists of DNA-based and non-DNA-based information. Also, not all DNA sequences are heritable, i.e., somatic cells do not pass on information to the next generation.
(b): I model the genome as the DNA sequences reconstructed (assembled) by experiments. It is true that the assembled genome contains errors, includes information in the somatic cells, and even lacks the DNA sequence in the germ cells. Nevertheless, because the genomes are now strings, they are easy to analyze by computers.
(c): In addition, researchers have improved the quality of the genome, i.e., they have revised the sequence over and over again to collect the information embedded in the DNAs.

families' histories are usually small. Finding the relation between genomes and disease is much harder than looking for a needle in a haystack.

Researchers have invented a method named *genome-wide association study* (GWAS) to solve these issues. Specifically, if they want to find a variation in genomes that correlates with a specific disease, say schizophrenia, then they collect genomic data from humans with the disease (called "case") and without the disease (called "controls") to find variations enriched in the cases. By using a large number of genomes from a large population, GWAS has revealed many associations between the genomes and the phenotypes ([142]).

The critical point of the GWAS studies is to search numerous variations in the genome without any prior bias to find correlations between variations and diseases previously missed. [3]

According to the review ([131]), the first GWAS study was in 2002 ([109]). It was on a genetically narrow Japanese population to find a relationship between SNPs and a cardinal disease. It has been 20 years since the first GWAS, and there are more than three thousand of GWAS studies, according to another review ([142]).

Notably, almost all genome-wide association studies depend on the genome sequence to find variations. In other words, it became possible only after we determined the genome.

Diversity among population

As we are human, we have an internal desire to explore ourselves – where do we come from? What are we? Where are we going? These questions are precisely what the famous artist Paul Gauguin tried to answer through his paintings and what some biologists have been trying to answer through genomics. They need to examine many human genomes because our genomes are different from each other, and one genome can not explain all of the biological diversity.

So far, The Human Genome Diversity Project ([20, 13]), the 1000 Genome project ([35, 10]), The HapMap project ([4, 38]), and The Human Pangenome Reference Consortium ([149, 75]) have examined the diversity of human genomes intensively. For example, The Human Genome Diversity Project collected and sequenced about 1000 genomes worldwide from 1991 to 2020. From these datasets, the authors measured genetic diversity quantitatively. Specifically, they found the genomes of populations in Africa, America, and Oceania have tens to hundreds of thousands of variations unique to each population.

As the genome sequences have been inherited from generation to generation, it seems possible to reveal the evolutionary trail we have taken. I will discuss this topic in the next section.

[3] Nowadays, GWAS usually picks one SNP for one genomic region with high linkage disequilibrium, making the number of statistical tests as small as possible.

Evolutionary analysis

Several years ago, I was watching an old movie. In that movie, one villain - the evil person - blamed another guy as a "Neanderthal." Interestingly enough, revolutionary biologists proved that his assertion is partially correct. In other words, they showed that some of the human genomes came from Neandertals.

Specifically, from 2006 to 2010, a team led by Svante Pääbo sequenced the DNA molecules extracted from three bones of Neanderthals ([44, 101]). They revealed that the genome of Neanderthals was closer to the non-African population than the African population. Combining other results, the authors proposed a striking scenario – Neandertals and modern humans had interbred and produced offspring after leaving Africa. [4]

After the publication of this paper, evolutionary biologists – "paleogenomics researchers" – have revealed the connection between modern humans and other Archaic humans such as Denisovan ([89]). These results are great examples of the power of genomics. The genomic sequence tells us something unbelievable (that we have to accept, nonetheless).

3D structure of DNA molecule

So far, I have argued that genome sequences are essential to elucidate the relationship between genome and diseases, the genetic diversity of us, and the evolutionary trail of humans. In addition, I give an application of genome sequences in molecular biology.

Usually, a human cell has 22 pairs of autosomes and two sex chromosomes. [5] If we extract these chromosomes and concatenate them, it will be as long as $2 \times 10^6 \mu$m (2 meters) ([117]). How does a human cell pack these extremely long molecules into its nucleus with a volume of 10 μm^3? In other words, what is the three-dimensional structure of the DNA in a cell?

To answer this question, in 2009, Erez Lieberman-Aiden proposed a method called Hi-C to reveal which part of the DNA molecules is close to another part in a genome-wide manner ([76]). In their method, they first fix the DNA molecules in the cell with formaldehyde. Then, they cut the DNA molecules by enzymes, mark the end of the DNA with biotin, and ligate both ends to glue the cross-linked fragments. Each ligated fragment consists of two DNA fragments close to each other in the cell. Finally, they collect these ligated fragments and determine the sequence of these DNA fragments.

The crucial point is that they can search these fragments' locations in the sample's genome. They determine two locations per fragment, corresponding to

[4] I feel it underrepresents their work. They spent twenty years developing a method to extract DNA from the bones. Bacteria heavily contaminated the extracted DNAs, and more than 90% of the DNAs were presumably derived from microbes that colonized the samples. The sequencer at that time only produced 5.3Gbp out of these extracted samples, corresponding to less than two-fold coverage. Albeit these difficulties, the authors revealed one of the great discoveries of the 00s'.

[5] Some cells, such as blood cells in humans, do not have nuclei and chromosomes.

the contacts of the DNA molecule in the cell.

The Hi-C method has revealed not only the three-dimensional structures of the DNA but also that these structures regulate gene expression. These structures change drastically and dynamically during the embryogenesis and development of individuals, opening another aspect of evolutional-developmental (Evo-Devo) biology ([95]).

Genomes are building blocks of our knowledge.

In this section, I examined the definitions of a genome and regarded the DNA sequences in a cell as the genome. Then, I briefly surveyed how researchers have used genomes to reveal new biological insights such as GWAS, populational diversity, evolutionary trails, and the three-dimensional structures of DNA molecules.

In the next section, I will explain how to reconstruct the genome as a string on computers.

1.2 Genome assembly problem

1.2.1 Abstract

A genome can be very long. For example, a human genome consists of 46 DNA molecules (chromosomes) that are roughly 6,000,000,000 characters in total. A human genome is too long to analyze by the eye, and we use computers to elucidate genomes. [6] To this end, we first need to observe the DNA molecules of the genome.

For now, the machine that determines the order of the bases in a DNA molecule, i.e., a *DNA sequencer*, can only sequence a tiny portion of the genome at once. These partial observations are called *reads* and are usually shorter than 1,000,000 bases even if we use the most advanced technologies. Thus, we need to glue these reads to reconstruct the DNA molecules as DNA strings on computers. This problem is called *genome assembly problem* and has been extensively studied so far.

Theoretically, the genome assembly problem is NP-hard, i.e., it is as hard as other well-known difficult problems such as the traveling salesperson problem and the vertex covering problem. Also, genome assembly is difficult in practice because there are DNA sequences that are longer than the reads and appear in the genome multiple times. These sequences are called "repeats", hindering an assembly program from estimating correct genomes from the reads.

Before the 2010s, the reads were very short (< 1 Kbp), and it was impossible to completely reconstruct large genomes such as a human genome in computers.

[6] Of course, we read our genomes through our bodies. Polymerases, transcription factors, histones, and other proteins in our cells read our genomes daily. When they finish reading, we die.

Instead, the researchers tried to reveal the entire set of genes and achieved fruitful results.

During the 2010s, the DNA sequencers improved to produce longer reads. The length exceeded several thousand bases, and it gradually became possible to generate longer and longer assemblies, aiming to reconstruct most of the genome sequences.

Toward the 2020s, the reads' length and accuracy continued to improve, and one manufacturer rolled out a DNA sequencer that could produce reads longer than 25Kbp with an accuracy of more than 99.9% reads called "HiFi" reads. Another company started selling a DNA sequencer that could constantly produce reads with hundreds of thousands of bases. Combined with other technologies, around 2021, the complete gapless human genome assembly was published.

As noted in the previous section, these assemblies are called *reference* genomes and have been fundamental data in biology. One of the famous uses of these references is representing a sample's genome based on them. For example, most of the genome of a person is almost the same as the human reference genomes, and we can approximate a sample's genome by single nucleotide variations (*SNVs*) and structural variations (*SVs*) on the reference. This representation has boosted the efficiency of the research on a populational scale, and some researchers analyzed more than 100,000 individuals.

Nonetheless, representing the genome of a sample by SNVs and SVs is an approximation, and the genome and the reference sequence can be very different. For example, in human genomes, the major histocompatibility complex (MHC) and leukocyte receptor complex (LRC) regions have so diverged among a population that there is no obvious way to compare these regions to the reference. Another example is the DNA molecule in the mitochondria in plants. These DNA molecules are called "mitochondrial genomes" and show high structural diversity even in the same species. The approximation based on SNVs and SVs is no longer valid in these cases, and we need another approach to elucidate these regions.

To improve this insufficient representation, I present `JTK` software to assemble these regions anew for each sample efficiently and accurately in Chapter 2.

1.2.2 What is genome assembly? – the formalization

The genome assembly determines the order of the four characters, `A`, `C`, `G`, and `T` of DNA molecules from the data produced by a DNA sequencer. This task has not been a trivial task for several reasons.

First, the observations can be erroneous, and we need to remove these errors from the input reads. Because we do not know the correct sequence *a priori*, removing errors is essentially an ill-defined problem.

Second, even if there are no errors, the technology to determine a DNA molecule's sequence only produces a partial observation of the molecule. For example, the best-selling DNA sequencer sold by Illumina outputs reads as short

as 150 base pairs. Thus, we need to stitch these reads together to reconstruct the original DNA sequences in computers, and we call this process *genome assembly*.

Although many researchers have proposed different formalizations, I will explain one of the most famous formalizations – the shortest superstring problem. For simplicity, I assume that there are no errors in the reads.

Definition 1.2.1 (Genomes and Reads)**.** Let $\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. They represent adenines, cytosines, guanines, and thymines, respectively. Also, let the set of DNA sequences with k-bp length be Σ^k. I call an element of Σ^k "k-mer." For example, the set of all 2-mer is

$$\Sigma^2 = \{\mathtt{AA}, \mathtt{AC}, \mathtt{AG}, \mathtt{AT}, \mathtt{CA}, \cdots, \mathtt{TG}, \mathtt{TT}\} \tag{1.1}$$

I represent all possible DNA sequences Σ^* as the union of these k-mers, i.e., $\Sigma^* = \bigcup_{k=0}^{\infty} \Sigma^k$. As a genome and a read are DNA sequences, I represent them as elements in Σ^*.

The problem of genome assembly is the reconstruction of the genome from a set of reads.

Although this problem *seems* to be rigorous, one critical issue is that we can not assess a solution to this problem. Specifically, suppose you want to assemble a genome from a set of reads. Then, one person approaches you and says, "I assembled this genome G' from these reads." Almost the same time, another person comes to your room and insists that he also assembled a genome G'' from the same set of reads, which would be better than G'. Because we do not know anything about the correct genome, G, it is hard to determine whether G'' is better than G'. To solve this issue, I regard a smaller genome as better.

Definition 1.2.2 (Shortest superstring)**.** Let R be a set of strings in Σ^*. Also, I say that a string G *contains* another string r when there are two integers i and j such that substrings of G from the i-th base to the j-th base are the same as r. Then, an assembly of the reads R is the string G containing every read in R. The shortest superstring of R is an assembly of G with minimum length.

This formalization defines what a good assembly is, and we can compare G' and G''.

Nonetheless, there are still two issues regarding this formalization.

First, there can be many assemblies that give the minimum length, and only one of them can correspond to the actual genome. Formally, let $\mathbb{G}(R)$ be the set of strings that contains every read in the input reads R. Then, the shortest superstring problem is the problem of finding one of the shortest genomes.

$$\arg\min_{G \in \mathbb{G}(R)} |G| \tag{1.2}$$

However, there can be many G that are minimum in length, and we can not determine which is the best assembly among them. For example, suppose we have four reads as follows:

$$R = \{\texttt{AT}, \texttt{TA}, \texttt{AG}, \texttt{GA}\} \tag{1.3}$$

One of the shortest superstrings of R is ATAGA. However, another string AGATA is also the shortest. From these reads R, there is no way to determine which assembly is better. Thus, solving the shortest superstring problem does not always give the correct DNA sequence.

In addition, the optimal solution does not always correspond to the actual DNA molecules in the cell. For example, suppose we have three reads, AA, AAA, and AAAA. The shortest superstring of these reads is G = AAAA because it contains these three reads and is minimum in length. Nonetheless, the actual genome may be G = AAAAAA. Thus, the solution to the shortest superstring problem can be shorter than the actual genome.

It is true that the shortest superstring problem has indeed motivated efficient and practical algorithms for genome assembly. It is also true that there are many other formalizations of the genome assembly problem ([92]).

Albeit, as shown in this section, solving a problem such as the shortest superstring problem does not always reconstruct the actual DNA sequence in a cell. In my opinion, the genome assembly problem is hard to formalize rigorously, and what is important is to find helpful tools or models to handle genome assembly in computer science with the data available at that time.

The following section will briefly summarize how genome assembly algorithms and DNA sequencers have progressed. Note that these summaries are neither faithful to the timeline nor a thorough description of what has happened. Rather, I highlight a few essential papers and techniques. Our goal is to make it easier to grasp the theme of this thesis by describing them.

1.2.3 Genome assemblies before 2010 – capturing all genes

DNA sequencers read DNA molecules

By 2010, researchers had invented many principles of DNA sequencing, and many DNA sequencers appeared. I compress the long and rich history of these DNA sequencers into two of them; capillary sequencers and Solexa's sequencers. Because Illumina acquired Solexa in 2007, I refer to Solexa's sequencer as "Illumina's" sequencers hereafter.

At that time, one of the standard DNA sequencing methods was based on polyacrylamide gel electrophoresis (PAGE). The idea is to use different types of bases (dideoxynucleotide) with fluorescent tags. Specifically, it uses the following three ingredients:

- Four types of deoxyribonucleic acid (dNTP: dATP, dCTP, dGTP, and dTTP).

- Four types of dideoxynucleotide triphosphate (ddNTP: ddATP, ddCTP, ddGTP, and ddTTP) with fluorescent tags.
- DNA polymerases, which replicate the DNA molecule using the eight types of the bases above.

In ddNTPs, the 3'-OH group at the dNTP is converted to H-group, which hinders polymerase from elongating the DNA molecules. By amplifying the DNA molecules by DNA polymerases with dNTP and ddNTP, we obtain partially amplified DNA fragments ended with ddNTP. We measure the lengths of these fragments in narrow capillaries using PAGE and determine the last base of these fragments by fluorescent tags. These two pieces of information are sufficient to determine the sequence of the DNA molecules of interest.

Although sequencers based on this methodology are still in work as easy-to-use DNA sequencers, it produces a small number of bases in one experiment. Given that the genome can be larger than 3 Gbp in a human genome, more efficient and inexpensive methods to sequence DNA molecules were needed.

In the mid-90s, Drs. Shankar Balasubramanian and David Klenerman invented a method to sequence a DNA molecule with two important methodologies - the bridge PCR and the reversible terminator.

Bridge PCR is a specialized polymerase chain reaction to amplify a DNA molecule. The difference from the usual PCR is that the bridge PCR fixes and accumulates the amplified DNA molecules near the original (template) DNAs. When tagging these amplified DNAs with fluorescent markers, the signals from these completely identical DNA molecules are so strong that they can be observed via optical systems, i.e., image sensors.

To sequence the DNA amplified by the bridge PCR base by base, reversible diterminators are used. They are similar to ddNTP in that they hinder the polymerases from elongating DNA, but we can remove this blocking by chemical reaction, i.e., this blocking is reversible. Thus, we can elongate the DNA by one base with reversible terminators, observe the added base by fluorescent, remove blocking on these bases, and amplify the next base.

Combining these two methodologies, Illumina developed a fast, efficient, and inexpensive DNA sequencer. It outperformed other sequencers and became a *de facto* standard within a decade.

Then, how did the researchers at that time use these sequencers and analyze the genomes of interest?

To determine the genome sequence, many researchers before the 2010s' first split the genome into 100 Kbp to 1 Mbp segments. Then, they inserted these fragments into yeast or bacteria to create yeast artificial chromosomes (YAC) ([98]) or bacterial artificial chromosomes (BAC)([97]). Finally, DNA sequencers determine the sequence of these fragments, and merging them provides the entire genome sequence. [7]

[7] There are methods called "pair-end" reads and "mate-pair" reads that were essential to the

Several other researchers thought they could simplify this procedure. They sheared the DNA molecules, sequenced these fragments directly, and glued the reads together to reconstruct the genome. See ([151]) and ([43]) for the conversations between the advocates of these two methods.

Regardless of the methods they used, the genome assembly during the 90s' and 00s' were far from perfect because the reads were short and the genomes were too large and complex. Nonetheless, the researchers did their best and revealed the order of genes in the genome, the regulation factors in the genome, and the evolutionary trail of the genomes. In the next section, I pick the human genome and the barley genomes as a showcase of the genomes during this era.

The draft human genomes said that we had 30,000 genes

Human genomes are particularly important for biomedical research since the genome sequence can possibly tell the biochemical compounds and the metabolic process in the cells. In addition, researchers at that time thought that the human genomes would reveal why we were so complex compared to other organisms, such as worms and flies.

Technological difficulties aside, there were two major concerns about the project to assemble a human genome before 2000. First, naysayers argued that it would take too much money and time to produce the complete assembly of 46 strings, which is 3,000,000,000 characters in total. Second, some biologists insisted that most of the sequences in a genome would be junk and that the project would be self-satisfactory for the scientists in the "big science."

Concerning the first issue, the methods of genome assembly and DNA sequencing had been improving beyond their expectations. In the publication of the first draft of the human genome, the consortium wrote that it took only fifteen months to assemble the 80% of the human genome they aimed to assemble ([70]).

Regarding the second issue, the first human draft genomes ([70, 144]) surprised the researchers somewhat ironically. In contrast to their expectations, they found 30,000 to 40,000 genes in the assembled genomes, which were only two or three-fold larger than those of a fly or a worm. Still, they greatly overestimated the number of genes in the genome. Currently, a human genome has about 20,000 genes ([1]). Also, the human genome consists of many non-coding regions, but the fraction was larger than most researchers thought. See the commentary by Birney ([14]) for a detailed explanation of what the first human genome told us.

Then, how complete was the draft version of the human genome?

In the ideal case, the assembly represents the entire genome perfectly, and there is one-to-one corresponding between the strings in the assembly and the chromosome in the genome. However, assemblies of eukaryotic genomes during

genome assembly at that time. However, I skip describing these methods for simplicity.

this period were highly fragmented, and the draft human genome was no exception.

To discuss the quality of the assembly, the average length of the sequences in the assembly is not a good metric because the assembly contained a lot of very short sequences at that time, and they lowered the average length. To avoid this deflation, I define the widely used metric called N50 as follows:

Definition 1.2.3 (N50)**.** The N50 is the largest length such that 50% of all nucleotides are contained in contigs of size longer than that.

Formally, suppose there are N sequences in an assembly, and the lengths of these sequences are $X = \{x_1, \cdots, x_N\}$. Also, without loss of generality, suppose these sequences are sorted in ascending order, i.e, $x_1 \leq x_2 \leq \cdots \leq x_N$. Then, there is an index $k \in 1, \cdots, N$ such that the total length of sequences longer than x_k is more than half of the total length. Namely,

$$\sum_{i=1}^{N} x_i \times \frac{50}{100} \leq \sum_{i=k}^{N} x_i \tag{1.4}$$

Let k^* be the maximum index that satisfies Eq. (1.4). N50 is x_{k^*}.

For example, suppose the lengths of the sequences in an assembly are $\{10, 10, 10, 10, 50, 60, 100\}$. The total length is 250 and the total length of sequences longer than 60 is 160. Since 160 is greater than $250 \times \frac{50}{100}$, the N50 of these sequences is 60.

The N50 of the first draft of the human genome was around 82 Kbp([70]), which was 34,000 times shorter than the entire genome, 3 Gbp. In the other organism, the N50s of the assemblies were much worse. Nonetheless, researchers tried their best to uncover the property of genomes as far as they could. The draft assembly of the barley genome is a good example of such an endeavor.

The barley genome - better than nothing

Barley is the firstest domesticated crop species in human history. It is important not only as a livestock feed but also as a human food because it has been hinted that soluble dietary fibers in barley reduce the risk of several diseases, including type II diabetes ([140]). Thus, for both agricultural and medical applications, researchers have been trying to elucidate the genome of barley.

However, assembling the barley genome was challenging because more than 80% of its genome is repetitive. Intuitively, if we pick a 1 Kbp sequence from the barley genome, the sequence appears in other regions in the genome with a probability of more than 80%. In addition, the DNA sequencers at that time could only produce reads shorter than 1 Kbp. The assemblers could not determine the location of these repetitive sequences in the barley genome. As a result, they

obtained only 1.9Gbp sequences from the 5.1Gbp genome ([88]). The N50 of this assembly was 1,425 bp, indicating that the assembly was highly fragmented.

Although the genome was far from ideal, instead of recovering the entire sequences of the genome, the authors searched genes in these short assemblies and collected messenger RNA sequences from different tissues. They define this integrated representation that provided most of the genes and their expression in the cell as the "gene space" of the barley genome. From this representation, they drew biological knowledge. For example, they found at least one-third of the genes of the barley genomes expressed differently from tissue to tissue.

In summary, the researchers have produced a valuable representation of the barley genome assembly that is far from ideal. Even if an assembly reveals only a tiny fraction of the entire genome, it is better than nothing. [8]

1.2.4 String graphs and *de Bruijn* graphs

Thus far, I have postponed the algorithms and software to carry out assembly from the reads produced by DNA sequencers. In this section, I briefly explain the algorithms in the genome assembly, which are the foundation of the presented software, `JTK`.

DNA sequencers produce many partial observations of the genome of interest. These observations, or *reads*, are sampled randomly from the entire genome. [9] As a result, it is possible that some locations are not covered by any reads, making "gaps" in the observations. Also, some reads overlap in the genome, making the data redundant. We must solve two issues to recover a genome from the reads.

- How many reads do we need to cover all the regions of a genome?
- How can we reduce the redundancy of the reads?

Concerning the first issue, Lander and Waterman gave a theory about DNA sequencing ([71]). One of the main theorems is that the number of reads passing over a specific position in the genome follows a Poisson distribution.

Theorem 1.2.1 (Lander-Waterman's statistics)**.** Let G be the genome size, N be the number of the reads, L be the length of the reads, and $NL/G \to C$ when $N \to \infty$ and $G \to \infty$.

Also, assume that the genome is circular, and I sample reads uniformly from the genome.

Then, the number of the reads at a specific position of the genome, denoted as X, follows the Poisson distribution $\mathrm{Pois}(X \mid \lambda = C)$ when $G \to \infty$ and $N \to \infty$.

[8] They published a new version of the barley genome in 2021, which accurately reconstructed most of the repeats in the genome ([86]).

[9] Strictly speaking, reads from DNA sequences are rarely sampled randomly. DNA sequencers usually have "preferences" regarding DNA sequences. Namely, some regions are hard to observe accurately by some sequencers.

In other words, given the total length of the reads $T = NL$, the probability to see x reads on a position on the genome is $\frac{x^{T/G}}{x!}\exp(-\frac{T}{G})$. See ([71]) for the proof.

Thus, the probability that there are no reads crossing at a specific position in the genome is $\text{Pois}(X = 1|\lambda = T/G) = \exp(-\frac{T}{G})$. For example, suppose we want to cover all the bases of the genome at least once in expectation, i.e., $G\exp(-\frac{T}{G}) < 1$. Then we have following equations:

$$\begin{aligned} G\exp(-\frac{T}{G}) &< 1 \\ \exp(-\frac{T}{G}) &< \frac{1}{G} \\ -\frac{T}{G} &< -\ln G \\ \frac{T}{G} &> \ln G \end{aligned}$$

Assuming a human genome with $G = 3 \times 10^9$, we have $\frac{T}{G} > 21.8$, indicating that we need $21 \times 3 = 63$ Gbp reads in total to cover all the bases in the genome with sufficient probability.

To handle the second issue, i.e., to reduce the redundancy of the reads, there are two algorithms invented by the end of 2010. They are string graphs and *de Bruijn* graphs and still comprise the core of the theory of genome assembly.

String graphs glues reads by overlaps

One of the most intuitive ways to reduce redundancy is to merge similar reads. For example, if we have two reads, x = `GGGTTTT` and y = `TTATCCC`, we can merge these two reads into `GGGTTATCCC` because there two reads are "aligned" as follows:

```
GGGTTTT
   TTATCCC
```

In other words, we have insertions of `GGG` at the first three bases, two matches of `T`, one mismatch of `A` vs. `T`, one match, and three insertions of `CCC`.

Nonetheless, it is not obvious to glue these two reads in this way because we have another way to glue the reads:

```
GGGTTTT
    TTATCCC
```

We need to rigorously define the criteria to glue reads by an alignment between two reads to implement this procedure as a software program. To this end, I first define the alignment between two reads.

(Note that if a definition becomes too long, I add a horizontal line to indicate the end of the definition. Also, I use the 0-based index system. In other words, I access the first element of an array A or a string r by A_0 or r_0.)

Definition 1.2.4 (Edit operations and alignments)**.** The set of *edit operation* is the set of four elements, and I denote it as $\{\mathtt{I}, \mathtt{D}, \mathtt{M}, \mathtt{X}\}$. Intuitively, `I`, `D`, `M`, and `X` correspond to the insertion, deletion, match, and mismatch in the "alignment" I am defining.

Let r and q be two DNA strings, i.e., two elements of Σ^*. Also, let an alignment operation A be a N-length array of $\{\mathtt{I}, \mathtt{D}, \mathtt{M}, \mathtt{X}\}$. I denote an alignment operation as $A = A_0, \cdots, A_{N-1}$. For example, suppose we have two reads $r = \mathtt{AACTG}$ and $q = \mathtt{AAAG}$, and $A = \mathtt{MMXDM}$.

The *alignment path* between r and q from A is a pair of string $\hat{r}$ and $\hat{q}$ defined by the following procedure:

1. We initialize $\hat{r}$ and $\hat{q}$ as the two empty strings.
2. We initialize $i = 0$ and $j = 0$. They point to the position of r and q. If i and j exceed the length of r or q, the alignment path is undefined.
3. For each edit operation $a = A_0, \cdots, A_{N-1}$,
   (a) If a is `M` or `X`, we push r_i to $\hat{r}$, q_j to $\hat{q}$, and increment both i and j by one.
   (b) If a is `D`, we push r_i to $\hat{r}$, $-$ to $\hat{q}$, and increment i by one.
   (c) Otherwise ,i.e., if a is `I`, we push $-$ to $\hat{r}$, q_j to $\hat{q}$, and increment j by one.

In the running example, $\hat{r} = \mathtt{AACTG}$ and $\hat{q} = \mathtt{AAA-G}$. $\hat{r}, \hat{q}$, and A are five-length arrays and "align" as follows:

```
AACTG
MMXDM
AAA-G
```

A is an *alignment* between r and q if the following four conditions are satisfied. These conditions ensure that the $\hat{r}$, $\hat{q}$ and A align as in the running example:

- The alignment path between r and q from A is defined.
- The number of `D`, `M` and `X` is equal to the length of r.
- The number of `I`, `M` and `X` is equal to the length of q.

- $\hat{r}_i$ is equal to $\hat{q}_i$ if and only if $A_i = \texttt{M}$.

In the running example, $\hat{r}_i$ is equal to $\hat{q}_i$ if and only if $i = 0, 1, 4$.

As noted in the previous example, there can be many alignments between two reads.[10] For example, suppose we have two reads, $x = \texttt{GGGTTTT}$ and $y = \texttt{TTATCCC}$. Then, the alignment $A = \texttt{DDDMMXMIII}$ gives the following alignment path.

$$\begin{array}{rcl} \hat{x} & = & \texttt{GGGTTTT---} \\ A & = & \texttt{DDDMMXMIII} \\ \hat{y} & = & \texttt{---TTATCCC} \end{array} \tag{1.5}$$

Another alignment $A = \texttt{DDDDMMXIIII}$ gives another alignment path.

$$\begin{array}{rcl} \hat{x} & = & \texttt{GGGTTTT----} \\ A & = & \texttt{DDDDMMXIIII} \\ \hat{y} & = & \texttt{----TTATCCC} \end{array} \tag{1.6}$$

To select an alignment from these possibly many choices, we need to evaluate the alignments between two reads quantitatively. To this end, I define the *alignment score*. Intuitively, an alignment has high alignment score when it contains many matches and few insertions, deletions, and mismatches. I will introduce four parameters M, I, D, and U to quantitively evaluate these operations.

Definition 1.2.5 (Alignment score)**.** Given an alignment A, let $\#(\texttt{I}, A), \#(\texttt{D}, A), \#(\texttt{X}, A)$, and $\#(\texttt{M}, A)$ be the number of `I`, `D`, `X`, and `M` in A, respectively. Also, let M, I, D, and U be four numbers representing the score of one match, insertion, deletion, or mismatch, respectively. Then, $\mathrm{Score}(A)$ is defined as follows:

$$\mathrm{Score}(A) = M \times \#(\texttt{M}, A) - I \times \#(\texttt{I}, A) - D \times \#(\texttt{D}, A) - U \times \#(\texttt{X}, A) \tag{1.7}$$

Using these definitions, I define the overlaps between two reads.

Definition 1.2.6 (Overlaps)**.** Given two reads r and q, the overlap E from r to q is the alignment A between the suffix of r and the prefix of q that gives the maximum score.

Formally, let $|r|$ and $|q|$ be the lengths of r and q, respectively. Also, let us denote the suffix of the read r from the i-th base as $r_{i\cdots} = r_i r_{i+1} \cdots r_{|r|-1}$ and the prefix of q up to the j-th base as $q_{\cdots j} = q_0 q_1 \cdots q_{j-1}$.

Then, let us define $\mathbb{A}(r_{i\cdots}, q_{\cdots j})$ as the set of all possible alignments between $r_{i\cdots}$ and $q_{\cdots j}$.

The overlap from r to q is defined as follows:

$$E(r \to q) = \arg_{A,i,j} \max_{i,j} \max_{A \in \mathbb{A}(r_{i\cdots}, q_{\cdots j})} \mathrm{Score}(A) \tag{1.8}$$

[10] I refer to the alignment operations and the alignment path as the alignment interchangeably.

For example, let r = ACGTTTT and q = TTATACG. Then, the alignment giving the maximum score is the one between $r_{3\cdots}$ = TTTT and $q_{\cdots 4}$ = TTAT:

```
TTTT
MMXM
TTAT
```

Note that usually we discard overlaps with low sequence identity. For example, there are 1% errors in Illumina's sequencer, and they are mainly substitution errors. Thus, assuming the errors occur independently and randomly the sequence identity between two reads originating from the same position in the genome should be around 98%.

Even though we must set the alignment parameters based on a dataset so that we can obtain meaningful overlaps, I set these parameters one just for simplicity. [11]

We have an exact algorithm to compute the overlaps between two reads (Algorithm 1). Intuitively, I denote the substring from the k-th base up to the i-th base of r as $r_{k\cdots i} = r_k \cdots r_{i-1}$. For given two reads r and q, the dynamic programming in Algorithm 1 compute a $(|r|+1) \times (|q|+1)$-dimensional matrix D where

$$D[i][j] = \max_{k=0,\cdots,i-1} \max_{A \in \mathbb{A}(r_{k\cdots i}, q_{\cdots j})} \mathrm{Score}(A) \tag{1.9}$$

We can compute the score of the overlap from r to q as $\max_j D[|r|][j]$ (Fig. 1.2 for a concrete example). Note that the running time of the overlap algorithm is $O(|r||q|)$.

How can we reconstruct the genome from overlaps between the reads? One obvious way is to glue two reads by the overlap. For example, we can glue GGGTTTT and TTATCCC by overlaps of length four to obtain a longer read, GGGTTTTCCC.

However, we need to sophisticate this naive approach because of the following two reasons.

First, a read can overlap more than two reads with different alignment scores. In this case, we need criteria to select one overlap from them.

Second, a read has a direction (the forward or reverse strand), and we do not know this strand information. This lack of information causes unintuitive overlaps. For example, suppose we have a genome G = AAACGTACGTGG, and a DNA sequencer produced the following two reads:

[11] Usually, we use different scoring scheme, i.e., Affine gap penalty, and required the length and the score of the overlap to be large.

		T	C	A	T	C	C	C
	0	-1	-2	-3	-4	-5	-6	-7
G	0	-1	-2	-3	-4	-5	-6	-7
G	0	-1	-2	-3	-4	-5	-6	-7
G	0	-1	-2	-3	-4	-5	-6	-7
T	0	1	0	-1	0	-1	-2	-3
C	0	-1	2	1	0	1	0	-1
T	0	1	-1	1	2	1	0	-1
T	0	1	0	0	2	1	0	-1

Figure 1.2: The example of the dynamic programming table between r = `GGGTCTT` (y-axis) and q = `TCATCCC` (x-axis). The gray cells indicate the overlaps.

Algorithm 1 Overlaps between two reads

Input: Two reads r and q
Output: Overlap E from r to q

Initialize an $(|r|+1) \times (|q|+1)$-dimensional matrix D with zero
for $j = 1, \cdots, |q|$ **:**
    $D[0][j] \leftarrow D[0][j-1] - 1$ ▷ Move by `I` operation
for $i = 1, \cdots, |r|$, $j = 1, \cdots, |q|$ **:**
    $M \leftarrow$ if $r_{i-1} = q_{j-1}$ 1 else -1 ▷ `M` or `X`
    Mat $\leftarrow D[i-1][j-1] + M$
    Del $\leftarrow D[i-1][j] - 1$
    Ins $\leftarrow D[i][j-1] - 1$
    $D[I][j] \leftarrow$ max (Mat, Del, Ins)
return $\max_{j=0,\cdots,|q|} D[|r|][j]$

$$r = \texttt{AAACGTA}$$
$$q = \texttt{CCTCGTAC}$$

Although these reads do not overlap at first glance, they do. Let us denote the reverse complement of the string q as $\bar{q}$, then we have the overlap as follows:

$$r = \texttt{AAACGTA}$$
$$\bar{q} = \quad\texttt{GTAGGAGG}$$

Thus, for each pair of reads r and q, we need to check the overlap from r to q and from r to $\bar{q}$. For example, if we have three reads r_1, r_2 and r_3, we check whether the following overlaps have high alignment scores:

- r_1 to r_2 and r_1 to $\bar{r_2}$
- r_1 to r_3 and r_1 to $\bar{r_3}$
- r_2 to r_1 and r_2 to $\bar{r_1}$
- r_2 to r_3 and r_2 to $\bar{r_3}$
- r_3 to r_1 and r_3 to $\bar{r_1}$
- r_3 to r_2 and r_3 to $\bar{r_2}$

A string graph is a useful data structure to handle these two issues ([93]):

Definition 1.2.7 (String graph)**.** Let R be a set of reads and E be a set of overlaps between R. Then, the string graph constructed from R and E consists of the set of nodes (V) and edges ($E \subseteq V \times V$) defined as follows.

First, $V = \{r_T \mid r \in R\} \cup \{r_H \mid r \in R\}$. Here H and T represent the "head" and "tail" of a read. For example, Fig. 1.3 shows four nodes corresponding to two reads (r and q).

To represent the forward direction, I add an edge $r_H \longrightarrow r_T$ for each read r. Similarly, I have $r_T \longrightarrow r_H$ for each r to represent the reverse direction.

To integrate an overlap $e \in E$ from a read r to q, I add two edges as follows:

- If the overlap is between r and q, i.e., both in the forward strand, add $r_T \longrightarrow q_H$ and $q_H \longrightarrow r_T$. Fig. 1.3 shows this case.
- Otherwise, i.e., q is reverse complemented, add $r_H \longrightarrow q_H$ and $q_H \longrightarrow r_H$.

Also, each edge representing an overlap has two integers, (l_r, l_q), indicating the length of the overlap of the two overlapping reads. A simple algorithm can glue two reads r and q by these integers. Namely, we pop l_r bases at the end of the r's sequence and append q's sequence.

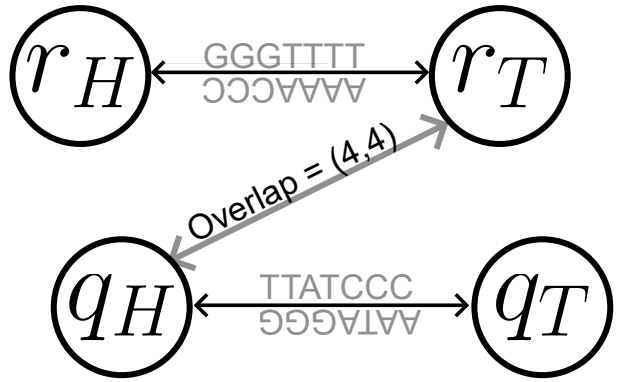

Figure 1.3: An example of a string graph.

To determine the overlap from a read r, i.e., to solve the first issue, a string graph has a transitive edge reduction algorithm to remove redundant edges. The algorithm runs in two steps (Fig. 1.4) and makes the graph much more simple.

1. Find a edge representing an overlap $v \longrightarrow u$. For example, suppose we selected $r_T \longrightarrow q_H$ in Fig. 1.4.
2. Remove all edges $v \longrightarrow w$ such that there is an edge from the other side of u to w. In Fig. 1.4, we remove $r_T \longrightarrow s_H$ because we have $q_T \longrightarrow s_H$.

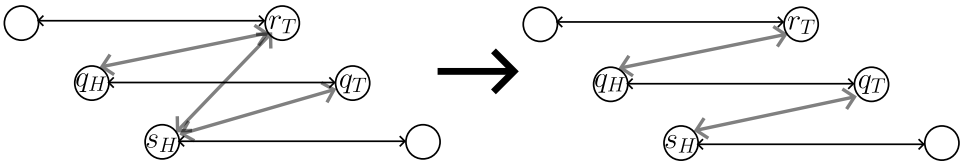

Figure 1.4: An example of transitive edge reduction.

Due to its simplicity, intuitive structure, and powerful performance, string graphs are used in some of the most common assemblers at that time, such as the `Celera` assembler ([94]).

Nonetheless, there are regions that a string graph could not assemble. For example, let d be a long, say 2,000 bp, DNA sequence. Then, consider a region in a genome consisting of three copies of d called *repeats* (d_1, d_2, and d_3, representing the same sequence):

$$G = \texttt{ACAG} \cdots d_1 \cdots d_2 \cdots d_3 \cdots \texttt{ACGTG}$$

If all the reads are shorter than 2,000 bp, a read derived from d_1 overlaps reads from not only d_1 but also d_2 and d_3 with the same alignment score. As a result, assemblers relying on string graphs can not select the right edge, usually breaking the assemblies around these repeats.

Also, the computation of overlap can take a too long time. Remember that we need more than $L = 21 \times 3 = 63$ Gbp reads to cover a human genome. Overlapping between these reads requires $O(L^2)$ time in total. Given that $63^2 \approx 4,000$, we must fill 4×10^{21} cells of the dynamic programming matrix in Algorithm 1. Usually, this is far more expensive than one can afford.

To improve the computational efficiency, there are at least two approaches. One is to assume that the error rates of the reads are low and use heuristics to approximate the overlaps. For example, Simpson and Durbin proposed a heuristic based on a suffix array to accelerate the computation of overlaps ([134]).

The other approach is to abandon string graphs and use another algorithm. This approach is called *de Bruijn* graph, and I will explain this method in the following subsection.

de Bruijn graphs find exact k-mer

In the 90s and 00s, the reads usually contained $p(=1)\%$ errors. If we picked a subsequence with the length of $l(=10)$ from these reads, the probability that the l-mer was free from sequencing errors was $(1-p)^l = 0.99^{10} \approx 0.9 = 90\%$.

Pavel Pevzner found in the 90s that all we need to do is to determine the order of these error-free subsequences of reads to assemble a genome ([113, 116]). Precisely, he proposed *de Bruijn* graphs defined as follows.

Definition 1.2.8 (*de Bruijn* graph)**.** Let R be a set of reads and k be an integer. As a working example, let the genome to be assembled be $G =$ `ACGTAGGAGGC`, $R = \{$`ACGTAG`, `TAGGAG`, `AGGAGGC`$\}$, and $k = 3$.

The *k-de Bruijn* graph from R consists of the nodes and edges defined as follows.

- The nodes V are k-mers, or the substrings with the length of k, in the reads. Formally, $V = \bigcup_{r \in R} \{r_{i \cdots i+k} | i = 0, \cdots, |r| - k\}$. In the running example, $V =$ {`ACG`, `CGT`, `GTA`, `TAG`, `AGG`, `GGA`, `GAG`, `GGC`}. Note that the redundant 3-mers such as `TAG` in the first and second reads appear only once in the nodes V.
- For each substring with the length of $k+1$ in the reads, I connect the node representing the k-length prefix to the node representing the k-length suffix. Formally, let us denote the two nodes v and u as k-length string, i.e., $v = v_0 \cdots v_{k-1}$ and $u = u_0 \cdots u_{k-1}$. Then, the edges E contains (v, u) if and only if $v_0 \cdots v_{k-1} u_{k-1}$ is a substring of some read in R.

The example of the *de Bruijn graph* in this example is depicted as Fig. 1.5.

Then, the problem of the genome assembly is to find a path without branching in this *de Bruijn* graph. In the example Fig. 1.5, we reconstruct the subsequence `ACGTAG` of G. In general, from the given reads R, we reconstruct a genome as the following procedure.

$$R = \{\texttt{ACGTAG}, \texttt{TAGGAG}, \texttt{AGGAGGC}\}$$

$$V = \{\ \texttt{ACG}, \texttt{CGT}, \texttt{GTA}, \texttt{TAG}, \texttt{AGG}, \texttt{GGA}, \texttt{GAG}, \texttt{GGC}\ \}$$

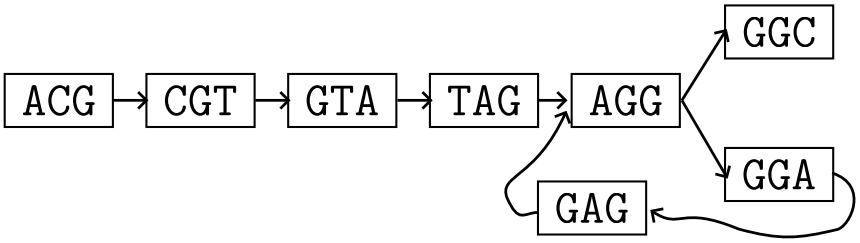

Figure 1.5: An example of the *de Bruijn* graph.

1. (Optional) Try to remove errors in the reads R.
2. Construct k-de Bruijn graph from the reads.
3. (Optional) Try to remove nodes (k-mers) containing sequencing errors.
4. Enumerate simple paths, i.e., paths that do not have branching.

When proposed in 2001 ([115]), the authors claimed the *de Bruijn* graph-based approach is better than the string graph-based approach because the problem on a graph is easier. Specifically, it had been said that in the string graph-based approach, we needed to solve a problem called Hamiltonian Path Problem, which everyone agreeded to be difficult. In contrast, in the *de Bruijn graph*-based method, the problem was to find a path that uses every edge exactly once.

For example, in the graph in Fig. 1.5, the path corresponding to the genome G is $\texttt{ACG} \to \texttt{CGT} \to \texttt{GTA} \to \texttt{TAG} \to \texttt{AGG} \to \texttt{GGA} \to \texttt{GAG} \to \texttt{AGG} \to \texttt{GGC}$, which uses all edges exactly once. This problem is known as the Eulerian path problem and is solved by a $O(|E|^2)$-time algorithm, which can be improved further. Compared to the string graph where obtaining the optimal assembly becomes intractable, the *de Bruijn* graph formalization provides us a problem that is tractable by usual computers.

Nevertheless, the de Bruijn graph-based approach is not a silver bullet. Here I point out three problems among several limitations of the de Bruijn graph.

First, like string graphs, *de Bruijn* graphs are not tolerant of repeat sequences. For example, the genome G in the running example (Fig. 1.5) has two occurrences of the 3-mer, AGG. This 3-mer makes a loop in the *de Bruijn* graph because there are two 3-mers, TAG and GAG, that connect to this node. Without any additional information, resolving these loops is generally impossible, breaking the assembled sequences at these loops.

Second, I point out that there can be multiple Eulerian paths in the *de Bruijn* graph. Of these paths, only one corresponds to the actual genome, and there is no way to find the correct Eulerian path. Intuitively, whenever we encounter a branch in a *de Bruijn* graph, we cannot determine which edge an Eulerian path should use first.

Third, because the *de Bruijn* graph does not take into account the similarity between k-mers, sequencing errors greatly affect assembly performance. In particular, errors within reads typically create k-mers that are not in the genome, making the graph more complicated. [12] This vulnerability to sequencing errors also makes the *de Bruijn* graph difficult to use with a large value of k. For example, if the error rate is 1%, we will typically see one error per 100 bp. Thus, using a large k, such as 200, would make almost all k-mers erroneous when the error rate is 1%.

To solve these problems on the *de Burijn* graph, researchers have invented many heuristics and software programs. They are ABySS ([135]), EULAR ([115]), SPAdes ([12]), and Velvet ([161]), to name just a few.

In my opinion, what makes the *de Bruijn* graph-based approach attractive is not the Eulerian path in the graph. As stated above, there can be several Eulerian paths, and we can not determine which path to select. Rather, what *de Bruijn* graph tells us is that the overlap computation is not mandatory. What we need to estimate is the set of k-mers and their order in the genome, both of which have nothing to do with the alignments.

Algorithms and DNA sequencers

In summary, I have very briefly explained the workflow of the genome assembly problem.

Although the two ideas - string graphs and *de Bruijn* graphs - were powerful tools for genome assembly, because the length of the reads was short, these assemblies could not produce contiguous assemblies around repeats. This problem - the repeat resolution - was recognized as early as 2004 ([114]), but bioinformatics researchers had to wait for the emergence of a new technology, the long-read sequencers, to tackle this problem.

1.2.5 Genome assemblies in the 2010s' – fight against repeats

The invention of the long-reads

The presence of repetitive sequences had hindered bioinformaticians from assembling the entire chromosome. If a DNA sequence longer than the reads occurs

[12] Sometimes an error in a read can create a k-mer that occurs in a different region of the genome.

more than twice in the genome, it is impossible to assemble it correctly and confidently by both string graphs and *de Bruijn* graphs. [13] Genomes usually contain tons of such repeats. As shown in the previous section, more than 80% of the barley genome consists of repetitive sequences. Because there was no way to overcome this issue through algorithmic improvements, bioinformaticians needed to wait for the progress of the DNA sequencers.

Also, the cost of the sequencing hindered the genome biologist from analyzing the entire genome. It cost more than 3 billion dollars at that time ([21]) to assemble just one human genome. Due to this high cost of DNA sequencing, the genome assembly project in the early 00s' tended to be an international project rather than a laboratory project that could be done with small grants.

To solve the second issue - the cost bottleneck - the National Human Genome Research Institute (NHGRI) started a project as known as the "1000$ genome project" in 2004 to encourage venture companies to develop inexpensive DNA sequencers. This project founded ninety-seven groups, and only two technologies survived: PacBio (formally Pacific Bioscience) and Oxford nanopore technologies. Interestingly, they provided new types of sequencing technology that can generate reads longer than 20,000 bp, which enabled bioinformaticians to find a way to resolve repetitive sequences. Here, I describe the principle of these sequencers very briefly.

The PacBio's DNA sequencer has a plate with many tiny wells called Zero Mode Waveguides ([65]). At the bottom of these wells, there are DNA polymerases. These DNA polymerases synthesize the DNA molecules in a sample with nucleoside triphosphates with fluorescent tags. Because the wells (Zero Mode Waveguides) are so narrow, when we light up these polymerases from the bottom of the wells, the DNA at the end of the synthesized DNA molecule exclusively absorbs the light and sheds fluorescent. We can record these fluorescent lights below the wells and determine the DNA sequences.

Because it can sequence the DNA as long as the polymerase synthesizes the DNA molecule, the length of a read is as long as several thousand to tens of thousands of bases.

The ONT's sequencer has a plate with many tiny protein pores called "nanopore" ([150], also see [18] for nanopore-based technology in general). The diameters of these pores are as narrow as DNA molecules. Also, there are ionic currents through the pores, and we can accurately measure these currents on each pore.

As the applied DNA molecule passes through the pores, the ionic current in the pore is blocked by the nucleotide in the pore. Because different nucleotides block the ionic flow differently, we can determine the base in the pore during the traversal of the DNA.

[13] If we chose a path at random, we might assemble the correct genome. However, it is due to mere luck.

Like PacBio's sequencer, ONT's sequencer can read DNA molecules as long as the activity of the pore is high. As a result, it can sequence as long as or even longer than 1,000,000 bp.

Interestingly, these two sequences not only produced longer reads than Illumina's reads but also had a common downside. The error rate of the reads was as high as 15%, and the variance of the error rate was also high. In the PacBio's or ONT's reads, some reads are 95% accurate while others are as accurate as 80%, making the average error rate 15%.

Researchers tried to handle these high error rates and leverage the length of these reads to make plenty of progress in genome assembly and comparative genomics. I pick the expansion of the available genomes and the assembly of the 32 Gbp genome of *Ambystoma mexicanum* (the axolotl) as examples during this period.

The pace of publication of assemblies has increased since 2010

The cost of DNA sequencing decreased, and long reads became available in 2010. Consequently, the process of genome assembly was simplified, became inexpensive, and significantly increased the number of the available assembly. Fig. 1.6 shows the number of the publication hit by searching "whole genome assembly" via PubMed (`https://pubmed.ncbi.nlm.nih.gov/?term=whole+genome+assembly`). Although this analysis is crude, it shows that the number of publications in 2020 was five times higher than in 2010.

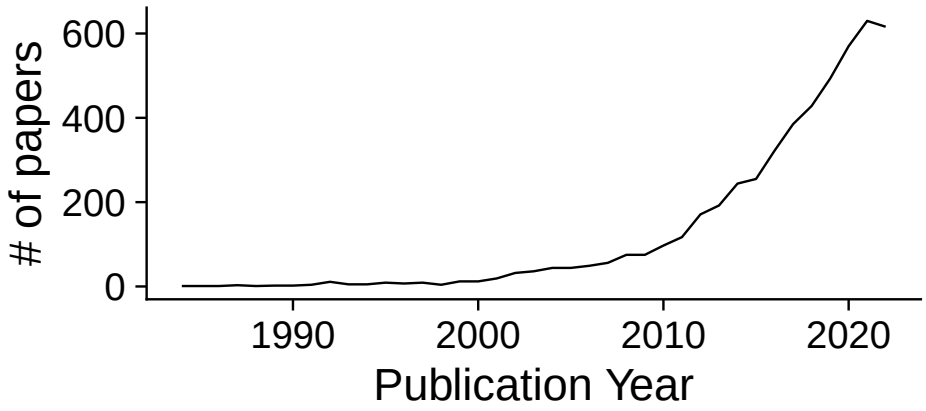

Figure 1.6: Relation between years and the number of publication hit by "whole genome assembly."

During this time, researchers began to realize that calculating the N50 value was not sufficient to assess the quality of an assembly for two reasons. First, they can trivially improve the N50 by simply concatenating the assembled sequences.

Second, the N50 and other metrics typically evaluate the contiguity of the assembly, and researchers wanted to evaluate the assembly in terms of biology, such as the gene content of the assembly.

To evaluate the quality of the genome in terms of gene content, it makes sense to check whether the assembled genome contains genes that are essential for development. For example, since the vertebrate body plan is highly conserved, all vertebrate genomes should contain Hox gene clusters.

One of the first papers to explore this idea was CEGMA ([110, 111]). It relied on the core eukaryotic genes (CEGs), which are thought to be highly conserved in low copy number in higher eukaryotes. CEGMA finds these CEGs in a newly assembled genome using alignment software and statistical models (pair hidden Markov models) to test whether the assembly had the genes it should have had.

Generalizing this idea, Simão *et.al.* proposed a method to check whether the assembled genome contains a set of genes that were found to be highly conserved and single-copy in the previously assembled genomes. It is called BUSCO (Benchmarking Universal Single-Copy Orthologs) and has been widely used to complement the N50 measure ([133]).

Today, a pipeline called QUST-LG integrates both sequence-centric measures (e.g., N50s or assembly sizes) and biological measures (BUSCO values) to provide a useful interface for assessing assembly quality.

Axolotol has a 32 Gbp genome, and researchers could assemble it

Ambystoma mexicanum, as known as the axolotl, is not only cute but also important species due to its ability to regenerate their lost legs ([146]). To investigate what genes and regulatory elements are essential for regeneration, the genome assembly of this species is necessary. Nonetheless, the assembly had been complicated because of its size. It is ten times larger than the human genome, comprising 32,000,000,000 base pairs in 14 chromosomes. Why is the genome so large? What kind of genomic element is responsible for this explosion of genome size?

To answer these questions, a dedicated assembler named `MARVEL` was developed ([103]). The authors sequenced the axolotl genome with around 2,000 sequencing experiments to obtain 1.5 Tbp datasets and assembled the genome with N50 = 200Kbp.

From the assembly, the authors revealed that the genome was filled with repetitive elements, especially transposons with long terminal repeats that expanded the length of the introns. If they had relied on the short read sequencers, this analysis could not have been possible because these regions are repetitive sequences, inhibiting a correct assembly via short reads.

The algorithm in the era of long-reads

After the long-reads emerged, it was expected that the assemblies would become more contiguous and accurate than ever. Nonetheless, we should not put new

wine in old bottles; we need new assemblers for the long reads. The problem was distinguishing repeats that were similar but not 100% identical.

Precisely, suppose we have a genome with two regions that are similar but not identical (black rectangles in Fig. 1.7). Also, suppose one read covers the first half of the inexact repeat, and another read covers the last half. Then, even if there are as many as 1% differences between repeats, the correct overlaps are hard to infer due to the high rate of sequencing errors.

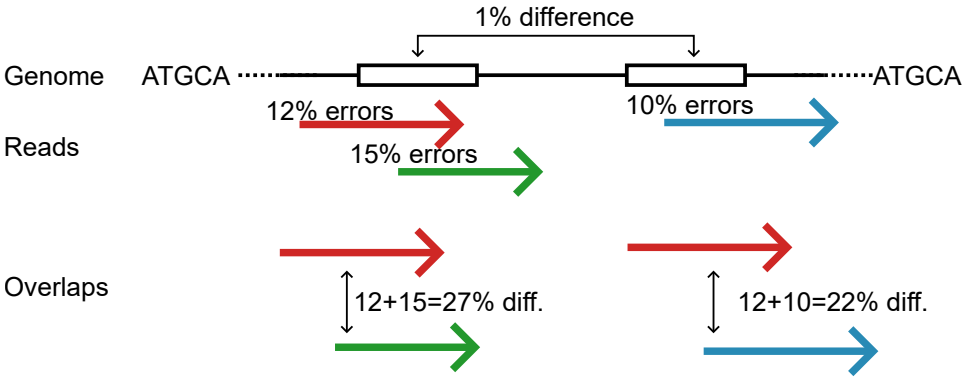

Figure 1.7: Under high error rates, even inexact repeats can not be resolved by read-vs-read overlaps.

To tackle this problem, on the one hand, assemblers based on string graphs employ two-step procedures. The initial step of these assemblers is to align the reads to each other to correct the erroneous bases in the reads. Then, the assemblers align these corrected reads again; Now, as the errors in the reads are corrected, the overlap will segregate inexact repeats.

Nonetheless, the argument above would become circular reasoning in the actual implementation. If we want to correct errors in the reads derived from the repetitive regions, we first need to have very accurate alignments between these reads. Again, the high error rate in the long reads prohibits us from obtaining these accurate overlaps of the reads derived from repetitive regions.

One approach to deal with this problem is to ignore all the repetitive sequences in the reads during overlap computation. `Canu` sophisticated this idea ([64]). Precisely, for each k-mer, such as 15-mer, `Canu` determine how repetitive the k-mers is in the datasets. Then, `Canu` reduces the match score of highly repetitive k-mers to obtain accurate overlaps. For example, if two reads share a repetitive 15-mer, `Canu`'s overlap module increases the alignment score less than 15 for this exact match to consider the repetitiveness of this 15-mer. [14] Based on these overlaps, it corrects the reads and goes to the second-round overlaps. This approach made a great success on the long-reads, especially PacBio's reads, and

[14] The actual computation of match score is complicated. Roughly speaking, they emploied the notion of "term frequency - inverse document frequency (TF-IDF)" in the natural language processing. See the original paper ([64]) for a further explanation.

it has been a *de facto* standard for genome assembly.

On the other hand, to use assemblers based on *de Bruijn* graphs, researchers needed to address two issues below.

First, suppose the error rate is 15%, and the errors are only substitutions. Then, a k-mer would not contain errors with probability $(1-0.15)^k = 0.85^k$. If we use k for Illumina's dataset, k is as large as 50, and this probability is much smaller than 0.1%. In other words, more than 99.9% 50-mers contains errors in the long reads. Many people thought that the probabilities was so high and it was impossible to use *de Bruijn* graphs with erroneous long-read sequencers. Also, *de Bruijn* graph split reads longer than 20,000 bp into k bp fragments, which seemed somewhat ridiculous at that time.

`Flye` ([63]) proved that these claims were invalid by using *de Bruijn* graphs to assemble genomes from long reads with two novel ideas ([63, 78]).

First, the authors noticed that they could segregate k-mers without errors from ones with errors based on the occurrences of the k-mers. Namely, if a k-mer, say `AAAT`, occurs many times in the dataset, it is unlikely that these occurrences are due to errors in similar k-mers such as `AACT` or `AAAG`. Rather, it is more likely that the k-mer is indeed in the genome, and these occurrences are the k-mer sequenced without any errors. Discarding k-mers occurs less than T times in the reads, say $k = 15$ and $T = 15$, `Flye` compute an accurate set of k-mers without errors. The authors of `Flye` call these k-mers *genomic*. Then, `Flye` constructs a variation of *de Bruijn* graph called *ABruijn* graph, where the nodes are these genomic k-mers, and edges are the connection of them in the long-reads.

Second, `Flye` solves repetitive regions after assembling the draft sequence from the *ABruijn* graph. Specifically, `Flye` first generated sequences without resolving repeats and concatenated these sequences. Although the generated assembly represents the sequence in the genome, the orders of the sequence between repeats are not correct in general. `Flye` employs another graph, named *repeat graphs* to fix the ordering of the sequences.

Using these two novel ideas, `Flye` assembler outputs the highly accurate assemblies from the erroneous long reads. [15]

Summary of the genome assembly in the 2010s

I started to study and research genome assembly algorithms in 2017. At that time, the reads were usually shorter than 15 Kbp ([73]). Sometimes we saw reads longer than 200 Kbp from ONT sequencers, but it had been suspected these reads were unreliable. In a few years, the length of the reads reached 100 Kbp, and even 1 Mbp reads were reported ([53]).

Using these long reads, researchers developed sophisticated algorithms, and biologists assembled the genomes of not only model organisms but also non-model organisms.

[15] Precisely, `Flye`'s contribution is the repeat graph, and the k-mer indexing is the contribution from the `ABruijn` assembler ([78]).

However, several regions could not be assembled. For example, there are regions that consist of short sequences (< 200 bp) repeating many times ($> 1,000$) in tandem, which is called "short tandem repeats."

$$G = \cdots \texttt{TTTT} \underbrace{\texttt{ACTACTACTACTACTACT}}_{\text{Short tandem repeats}} \texttt{GGGGG} \cdots$$

To see the reason why these regions are hard to assemble, suppose we have two reads:

$$r_1 = \texttt{TTTTACTACTACT}$$
$$r_2 = \texttt{ACTACTACTGGGGG}$$

The best overlap between these two reads is as follows:

```
TTTTACTACTACT
    ACTACTACTGGGGG
```

This overlap produces `TTTTACTACTACTGGGG`. However, it underestimates the number of `ACT` in the actual genome. To correctly assemble this repeat, we need reads spanning these repetitive regions by overlaps between reads ([160]). [16] Also, because of the high error rate, these short tandem repeats were difficult to assemble even when there were differences among the regions.

To fully assemble these regions, researchers needed to wait until 2019, when PacBio rebranded their sequencing methodology called "HiFi", and ONT's sequencers bacame able to produce reads longer than 500 Kbp constantly.

1.2.6 Genome assembly after 2020 - toward gapless assemblies

PacBio "HiFi" reads and ONT ultra-long reads

By the end of 2019, long-read technologies became commonplace, and many researchers assembled sequences longer than 1Mbp accurately. Also, they could use Hi-C ([76]) method to infer the distance between the assembled sequence in the genome in the cell, providing a way to order the sequences into each chromosome. As a result, chromosome-scale assemblies became available for non-model organisms.

Nonetheless, these genomes usually contain sequences repeating thousands of times called *short tandem repeats*. Also, large sequences sometimes duplicate in

[16] Of course, there is another problem; there are too many possible alignments. However, we do not argue the problem related to efficiency, i.e., running time, too deeply. Also, the length of the short tandem repeats can be estimated separately by counting the k-mers comprising the repeats.

the genome, making highly similar regions called segmental duplications. These two types of sequences were hard to assemble due to the errors in the long reads. To correctly assemble these regions, researchers thought that reads should be longer than 1,000,000 bp to span all the short tandem repeats, and reads should be as accurate as 99.9% to segregate all the segmental duplications. Surprisingly, sequencing technologies have these requirements satisfied.

First, around 2020, PacBio rebranded their circular consensus sequence (CCS) product into "HiFi" reads. [17] In the "HiFi" technology, the sequencer reads the same DNA molecule several times as in the previous erroneous technology. By aligning these erroneous reads, the "HiFi" technology locates and corrects errors to achieve more than 99.9% accuracy.

The high accuracy of the HiFi reads made the assembly process easier and more accurate than ever. Because there were almost no errors, assemblers based on string graphs could use stringent overlaps between the HiFi reads. `HiCanu` even used overlaps with 100% identical, i.e., overlaps without any mismatches, deletions, and insertions to construct the string graph. Also, *de Bruijn* graph-based assemblers such as `LJA` ([11]) and `Verkko` ([122]) used large k on the HiFi datasets, such as $k = 1{,}500$ or even as large as 5,000. As a result, they could segregate almost all the repetitive sequences in the genome whenever there are variations among the repeats, only leaving repetitive regions, such as very large short tandem repeats, to remain unassembled.

To solve the remaining long and exact repeats, there were two approaches. First, ONT provided very long reads called "ultra-long" reads longer than 200 Kbp. [18] Assemblers such as `Verkko` could infer the correct sequence by aligning these ultra-long reads onto the assemblies produced by the HiFi reads.

The other way was to use Hi-C reads. The Hi-C method provides pairs of sequences that are close to each other in the genome, it also tells how to solve the remaining repeats in the assemblies. `HiFiASM` is software that can use this information to generate very contiguous assemblies.

In summary, the HiFi reads from PacBio sequencers solved almost all the sequences in the genome, and ONT ultra-long reads or Hi-C datasets complemented the assembly. The most significant progress in this period was the long-awaited completion of the human genome – the telomere-to-telomere assembly of a human genome (T2T-CHM13) ([104]).

The complete genome of a human genome

Around 2003, the Human Genome Project officially announced that they sequenced and determined the human genome ([70]). The assembly has provided

[17] Technically, "HiFi" reads refer to the reads produced by the CCS method with more than 99% accuracy (https://www.pacb.com/technology/hifi-sequencing/how-it-works/).

[18] Different researchers used the term "ultra-long reads" in different meanings. In this thesis, I refer to the ONT reads longer than 200 Kbp as "ultra-long" reads.

fundamental knowledge of the following research, such as GWAS ([109]), and the team has kept updating the genome constantly.

Nonetheless, the assembly contained only **99**% of the **euchromatic** region of the human genomes, and the team has left the remaining 1% of the euchromatic region and the **heterochromatic** regions in the human genome unassembled. Most of these unassembled regions consisted of segmental duplications and tandem repeats, which were hard to assemble then.

To uncover the sequences of these unassembled regions, researchers started a project named "telomere-to-telomere project" intending to construct the gapless haploid assembly of the human genome. In this project, there were three key points.

First, a usual human genome in a cell consists of 23 pairs of autosomes and one pair of sex chromosomes. The differences between these pairs can be significant and make it challenging to construct a complete *haploid* assembly of the human genome. To avoid this issue, they used a sample of a complete hydatidiform male (CHM), where both pairs of chromosomes were derived from the father, i.e., sperm. Because this sample did not have any chromosomes from the maternal genome, there were almost no differences between homologous chromosomes in this sample, which made the sample virtually haploid.

Second, to find variations embedded inside simple repeats and segmental duplications, they used PacBio's HiFi reads. To maximize the accuracy, they further polished these HiFi reads and constructed a string graph from overlaps with exact matches, i.e., 100% identities. [19] This string graph separated almost all the repeats in the genome but left several segmental duplications and tandem repeats tangled in the graph.

Lastly, they used ONT ultra-long reads to assemble these segmental duplications and tandem repeats. Precisely, they defined k-mers corresponding to the variation inside segmental duplications or tandem repeats, located the ultra-long ONT reads in the string graph by using these k-mers, and resolved tangles in the graph one by one.

Although the authors needed to carry out intensive manual curations, they finally completed their objective – the complete human genome assembly (T2T-CHM13) ([104]).

The reference sequence improves every research using the previous incomplete assembly. For example, the assembly uncovered the 40-80 Mbp region in chromosome 9 in the human genome, which was a centromeric repeat and had not been fully represented in the reference genomes. Also, the researchers found the modifications of histones in the large segmental duplications in the human genome ([3]).

[19] Note that they also converted consecutive runs of the same bases into one base, e.g., converted `TGAAACCA` into `TGACA`. This "compression" of the homopolymers further reduced the errors.

The assembly has not ended – toward "personalized" genomes

The impact of the complete assembly of the human genome goes beyond changing the biological interpretation of the human genome. The assembly implies that any genomes that are easier to assemble than the human genome can be assembled by HiFi, ONT, and Hi-C reads.

Nonetheless, the problem of genome assembly has not finished yet. Even though we have the high-quality reference genome of a human, the genomes of each individual, such as you or me, are different from the reference. These differences can be so significant that the reference is insufficient to represent the sample's genome. In the next section, I will discuss this problem in detail.

1.3 The theme of this thesis

1.3.1 Abstract

To analyze an individual's genome, we usually have one assumption. The assumption is that the genome of a sample is so similar to the reference that we can represent it as a set of single nucleotide variants (SNVs) and structural variants (SVs). Assuming that, we determine these SNVs and SVs through sequencing technologies, such as Illumina's short reads, and analyze the genome by efficient algorithm on these SNVs and SVs.

Then, what do we miss by approximating the genome this way, and how is it harmful?

In this thesis, I pick two examples where the approximation does not work well. Specifically, in plant cells, the genomes in the mitochondria recombine frequently, making many structural variations. Also, in human genomes, the major histocompatibility complex (MHC) and leukocyte receptor complex (LRC) regions are so variable that we can not align the region in a sample to the reference genome, e.g., T2T-CHM13. The mitogenome of plants can have a gene inhibiting the creation of mature pollen, and the MHC region is the most frequently reported region in GWAS, making these regions agriculturally and medically important.

To fully represent these regions, it is necessary to assemble them in *de novo* for each sample. Moreover, when we aim to uncover the genomic landscape in several samples or on a population scale, it is desirable to assemble these regions with minimum sequencing requirements.

To this end, I develop a genome assembler focusing on these hard-to-assemble regions. The software, `JTK`, produced fully assembled genomes for the mitogenomes of the model plants, i.e., *Arabidopsis thaliana*, and diploid genomes of two human samples.

1.3.2 References and variants

In 2007, some researchers were so ambitious that they started a project to read the genome of 1,000 people worldwide ([35]). Their first assumption was that the DNA sequencer would produce six Gbp (6,000,000,000 bp) per person, and the total data would be as large as six tera-base pairs. However, increasing sequencing capacity forced them to change their mind. Within five years, the DNA sequencers could generate 80 Gbp per sample and produce more than 450 tera-bases. The issues were not only how to store these overwhelming amounts of datasets but also how to summarize these bases for downstream analyses.

One approach is compressing the sequencing data by assembling the samples' genomes. Because the size of a usual human genome is as large as three giga-base pairs, this compression would keep the project as-is, i.e., one 6 TB hard drive would be enough. [20] However, genome assembly was challenging at that time. Moreover, compressing the whole genome sequencing into assemblies squishes all heterozygous variations between homologous chromosomes, i.e., it erases the difference between the chromosomes derived from the father and ones from the mother. Thus, the assembly-based method seemed to remove too much information from the dataset.

Instead, they represented the genome as different from the reference genome. Moreover, they thought the following approximation held (Fig. 1.8):

> The genome of a sample is the set of single nucleotide variations (SNVs) and structural variations (SVs) on the reference.

Specifically, they aligned the reads from a sample's genome to the reference genome. Using these alignments, they found variations between the sample and the reference and consider them as the representation of the sample's genome. This approach includes the heterozygous and homozygous variations, avoids genome assembly, and represents the genome in very compact files.

Numerically, human genomes have around 0.1% differences compared to the reference sequence. To represent one variation, we need to specify the position and the type of variation, i.e., substitutions or insertions for four bases or deletion. As a human genome consists of roughly 3×10^9, we need $\log_2(3 \times 10^9) < 32$ bit to express a position in the reference genome. Also, there are nine types of variation, requiring at most $\log_2 9 < 4$ bit. Thus, representing one genome requires the following bits:

$$\underbrace{\left(\log_2(3 \times 10^9) + \log_2 9\right)}_{\text{Bits to represent a variant}} \times \underbrace{3 \times 10^9}_{\text{Genome size}} \times \underbrace{0.001}_{\text{\% difference}}$$
$$\approx 100,000,000 = 100\text{MB}$$

[20] Yes. An individual's genome is as large as **six** giga-base pairs, considering two homologous chromosomes.

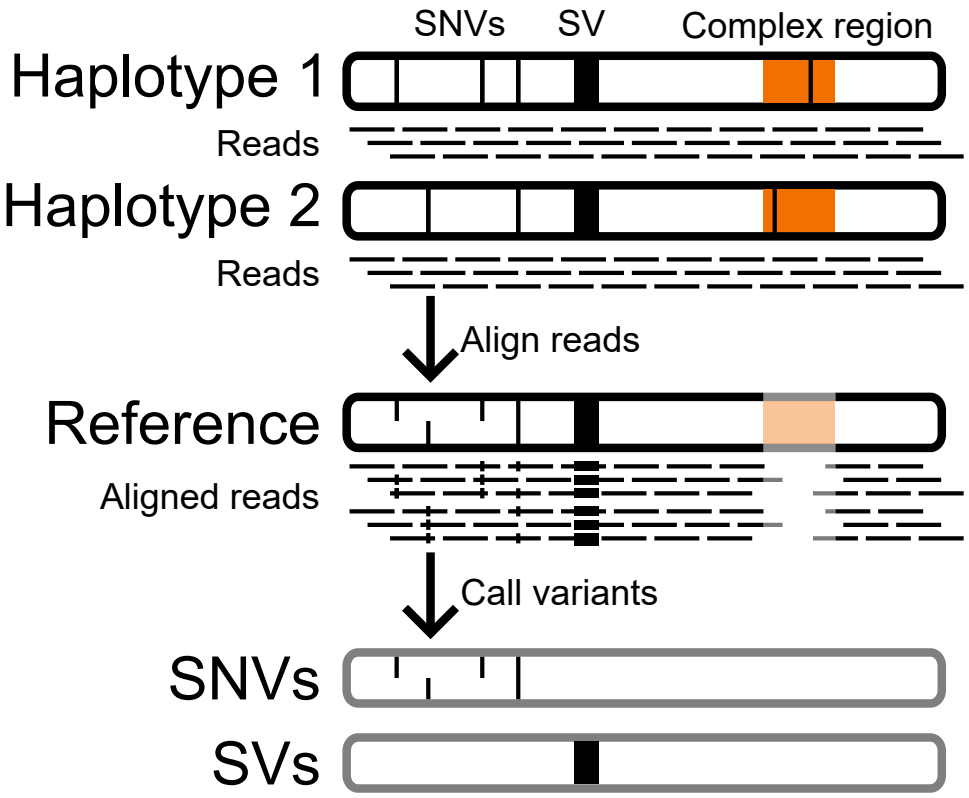

Figure 1.8: Schematic illustration of the representation of a genome by SNPs and SVs. There are one homozygous variant, three heterozygous variants in the sample's genome depicted as a horizontal line. There is also a homozygous SV illustrated as a black rectangle. In addition, there is one complex or highly diverged region colored orange, which can not be represented by the SNV and SV-based approach.

Therefore, representing 1,000 people requires only one 100 GB HDD, which is around 2,000 times more efficient than storing the raw DNA sequences.

The researchers extended this approach to include structural variations and implemented them as the variant call format (VCF) [27] in 2011. Since then, many researchers have written software programs producing or processing VCF files to analyze genomes. The most intensive studies so far are UK biobank ([138]) and GnomAD ([59]), which consist of 500,000 and 140,000 participants, respectively. Through this large amount of samples, the researchers have been connecting genomics and populational genetics to biomedical applications such as drug discovery ([152]). [21]

Nonetheless, the efficiency of VCF relies on the **assumption** that the differences between the genome of a sample and the reference sequence are single nucleotide variations (SNVs) and structural variations (SVs). Then, in what situation does this assumption become inappropriate? In the following two sections, I will explain two examples where the assumption does not hold.

1.3.3 Mitochondrial genomes in plants

Mitochondria is one of the organelles in the cell, and they have their DNA molecules. Researchers call these DNA molecules mitochondrial genomes or *mitogenomes* because these DNA molecules presumably arose by specific series of endosymbiosis from free-living bacteria ([125]). [22]

Mitogenomes have evolved, and those in the higher plants have distinctive features to those in animals. First, the plant mitogenomes are about ten to hundred times large than mitogenomes in vertebrates ([45]), comprising several hundred kilo-bases to several mega-bases. Also, mitochondria have a gene that inhibits an individual from creating mature pollen, which is called cytoplasmic male sterility (CMS) ([47]). Lastly and notably, the mitogenomes of plants recombine frequently, making the structure different from each other ([45]). Due to these recombinations, the structures of the mitogenome are different even among the different strains of the same species. For example, a previous study unveiled that the mitogenomes of the three strains of the model species, *Arabidopsis thaliana*, have different structures compared to the reference genome([143]).

These large-scale recombinations make it challenging to represent the mitogenome as the SVs on the reference genome. To fully present the mitogenomes of plants, we need the assembly of them instead of finding SNPs and SVs in the datasets.

[21] Funny enough, in AGBT2022, some participants complained that the VCF did not scale anymore. They were trying to analyze hundreds of thousands of people in a cloud platform as efficiently as possible.

[22] As Sagan admitted in her seminal paper, she was not the first one to **hypothsize** that the mitochondria were once free-living bacteria. Instead, her contribution was collecting the evidence for the hypothesis from the literature and making the hypothesis understood by everyone, which makes her work timeless.

1.3.4 Diploid genomes of humans

We have different genomes from each other. Usually, as argued above, we can represent these differences as SNPs and SVs on the reference. However, in some cases, these differences are so significant that it requires reliable methods to analyze these regions.

One example is the major histocompatibility complex (MHC) region in human genomes. This region contains a high diversity, many alleles among the human population, and plays an essential role in the body's immune system. For example, in the transplantation of organs such as livers, it is necessary to check whether the donor (the one who gives the organ) and the recipient (the one who receives the organ) share the same type of MHC region in their genome. If they have different alleles in the MHC region, the immune system of the recipient would consider the transplanted organ as "non-self" and reject it ([141]). Also, the variations in the MHC region are "usual suspects" in the genome-wide association studies and have attracted many researchers ([72]).

Nonetheless, this region's divergence among a population is so significant that SNVs and SVs are insufficient to represent them. Also, because we have two homologous pairs for each chromosome, we need to assemble these two haplotypes separately to elucidate the differences.

1.3.5 Genome assembly by erroneous long reads

To efficiently represent and analyze the genomes, we have represented genomes as a set of single nucleotide variants and structural variations. This approach does reduce the data requirements per genome, making it easy to develop software programs to analyze the genomes on a massive scale.

Nonetheless, this approach approximates the accurate picture of the genome, and we need to complement the representation in the complex regions such as the mitogenome of plants and the MHC region in human genomes. This is the first theme of this thesis. In other words,

A genome is more than a set of SNPs and SVs on a reference.

To this end, one approach is to assemble the genome *de novo* for each sample. As we successfully assembled the human genome, it might seem an easy task to carry out.

However, many essential factors contributed to the gapless human assembly, and some of these factors can be unavailable. Specifically, the complete human genomes relied on the CHM sample to ignore the difference between haplotypes, but that difference is exactly what we want to reconstruct. Also, they sequenced high-coverage datasets, including HiFi reads, ultra-long reads, and Hi-C datasets with more than 100-fold coverage. Although the throughput of the DNA sequencer has been improving, it is desirable to assemble a genome depending on single sequencing technology, i.e., either a PacBio sequencer or an ONT

sequencer. Lastly, manual curation was necessary to fully assemble the region, which should be removed whenever possible.

To fulfill this objective, we need to handle the errors in the reads, which is the second topic of this thesis. Namely,

Genome assembly is possible on erroneous long-reads.

1.3.6 Unifying strings graphs and *de Bruijn* graphs into *chunk graphs*

So far, I have argued that the representation of a genome through variants on the reference sequence is insufficient in the cases such as plant mitogenomes or the human MHC region. As a complementary approach, I aim to assemble the genomes in these cases *de novo*. In addition, it is desirable to assemble these genomes with minimum sequencing requirements to scale the investigations on these genomes.

One of the main issues we face when we want to assemble the genome with erroneous long reads is that the difficulty of assembly differs from region to region. Specifically, suppose we want to assemble the MHC region in the human genome in the diploid resolution. The differences between the two haplotypes vary dramatically in this region. Some regions in MHC have very similar (or even identical) haplotypes, while others have so diverged that we can not have a confident alignment between haplotypes. Handling both cases simultaneously is difficult by either string graphs or *de Bruijn* graph.

String graphs rely on the overlaps between the reads. There are two sources of base-level errors in these overlaps. One is sequencing errors between two reads derived from the same haplotype. The other is actual variations between a highly diverged region in MHC. As long as we rely on pairwise overlaps between the reads, it is difficult to separate true overlaps from false ones. [23]

de Bruijn graphs rely on the k-mers without sequencing errors in the long reads. However, it is challenging to obtain error-free k-mers from the reads due to the high error rates in the reads. Although `Flye` ([63]) and `wtdbg2` ([124]) try to solve this problem by dedicated and sophisticated algorithms, these approaches can not separate homologous chromosomes, especially when the similarity between the haplotypes is nearly 100%.

To solve this issue, I unify the string graph-based approach and the *de Bruijn* graph-based approach. To grasp the intuition, **suppose** we have erroneous long reads R and have assembled N possibly overlapping subsequences from the genomes, denoted by $G_1, \cdots, G_N$ from the taget genome G. I call them *chunks* (Fig. 1.9b). Then, I can compute the following two alignments (Fig. 1.9c).

- The **overlaps** between $G_1, \cdots, G_N$.

[23] If we use multiple sequence alignments among the reads, there is a way to distinguish these two. `HiFiASM` ([22]) is software to do that, and out software (`JTK`) does essentially the same thing.

- The **alignments** between the reads R and $G_1, \cdots, G_N$.

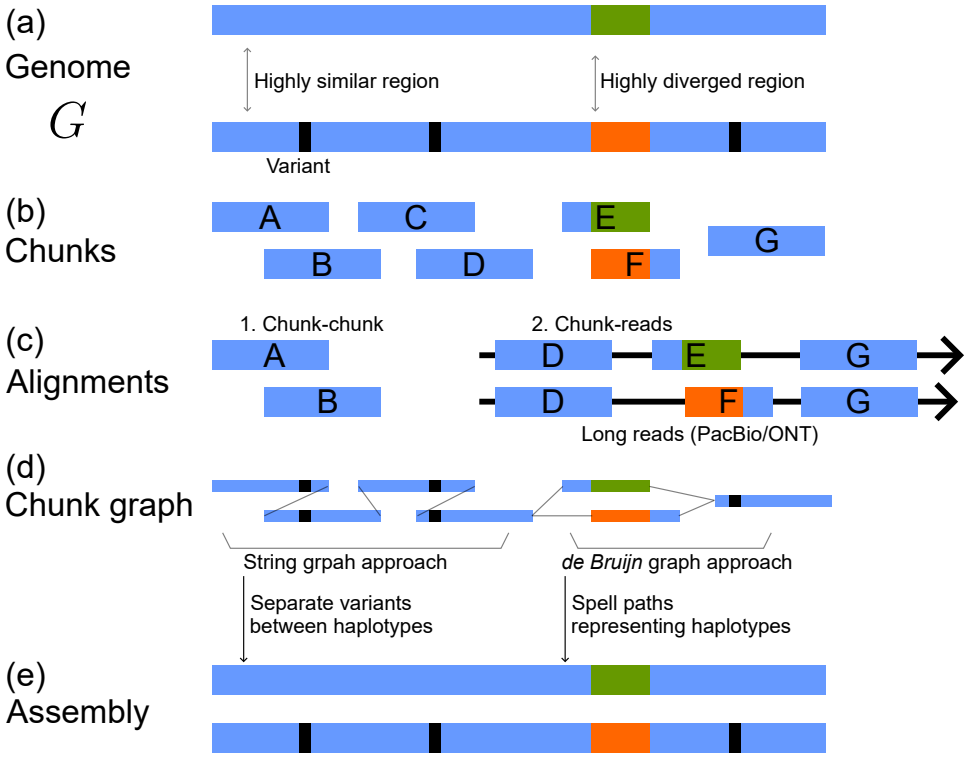

Figure 1.9: Unifying string graphs and *de Bruijn* graphs to assemble highly diverged regions and highly similar regions. (a): The genome has highly similar and diverged regions. (b): Assume that we have sampled several sub-regions (chunks) from the genome. These chunks can overlap, and there can be gaps between the chunks. (c): We have two types of alignments between the chunks and the reads. (d): A chunk graph can be created by merging these two types of alignments obtained in (c). (e): Separating homologous chromosomes and traversing haplotype-specific paths in the chunk graph produces the correct assembly graph.

By using the former alignments, I can construct a string graph between these N subsequences (1.9d, left). The string graph serves as a "reference" sequence on the region with low divergence, providing a way to reconstruct diploid genomes in these regions.

On the other hand, the latter alignments tell the connections between these N subsequences in the target region. By using these connections, I construct a graph where the nodes are these N subsequences and the edges are the connections. This graph is similar to a *de Bruijn* graph and assembles the target region with high sequence divergence (1.9d, right).

This approach seems to solve both highly diverged regions and highly similar regions in the genome (1.9e). Nonetheless, there is one critical flaw in this

argument. We never know the subregions $G_1, \cdots, G_N$ *a priori*.

In the next section, I present my software, `JTK`, to carry out the assembly from erroneous long-reads by refining this idea. Note that this approach has been hinted at by previous literatures ([84, 29]). Nonetheless, one paper ([29]) aims to assemble the transcriptomes, not the genome, of a sample and the other ([84]) does not work well on the real datasets, as shown in Chapter 2.

Chapter 2

JTK – regional genome assembler

Abstract

Genome assembly, the process of reconstructing the genome of a sample, is a fundamental process of genome biology because it gives us the entire landscape of the genome.

When the DNA sequencers produced short ($< 1,000$) reads, assembling the entire chromosome in one sequence was impossible due to repetitive sequences in the genome. However, now the DNA sequencers can produce reads longer than 500 kbp constantly, and it has become possible to assemble the haploid chromosome by a sufficiently large amount of datasets and manual curations. Further, it has been hinted that assembling the homologous chromosome separately is possible by combining various sequencing technologies. Nonetheless, it is still unclear whether or not we can assemble a genome in diploid resolution with single sequencing technology (i.e., PacBio or ONT) with moderate coverage (e.g., ~60-fold).

Here, I developed a *de novo* assembly program named `JTK` to assemble a megabase-scale region in a genome in haploid or diploid resolution from ONT or PacBio datasets. It first randomly samples kilobase-scale sequences (called "chunks") from the long reads, phases variants found on them, and produces two haplotypes. The novel idea of `JTK` is to utilize chunks to capture SNVs and SVs simultaneously.

The phasing module in `JTK` showed that it could phase variants even when there was only one SNV in a region by using 60-fold erroneous reads. On the synthetic haplotypes, `JTK` outperformed other ONT-based assemblers such as `PECAT` ([100]) or `Phasebook` ([84]) in terms of consensus and phasing accuracy. I also observed that `JTK` could handle large ($\sim$ 1 Mbp) and highly identical ($>$ 99%) segmental duplications.

Further, it is possible to reveal the diversity among the mitogenomes in plants

and elucidate the haplotype diversity among human genomes, which are the topics of the rest of this thesis.

Code availability

The source code of JTK is available at https://github.com/ban-m/jtk. The datasets and the results in this section are available at https://mlab.cb.k.u-tokyo.ac.jp/~ban-m/dthesis/jtk.tar.gz. These files are also available upon request (banmasutani@gmail.com).

2.1 Introduction

The problem of genome assembly has changed its core problem from revealing the entire set of genes to obtaining longer contigs and then reconstructing the entire chromosomes. Nowadays, creating chromosome-scale reference assemblies is possible with a sufficient dataset and manual curation. Using these references, researchers represent the genome of a sample as a set of SNPs and SVs on the reference.

For example, the GnomAD project and the UK biobank project ([138]) sequenced more than 100,000 human samples. These genomes have been represented based on SNPs and SVs and used to validate the candidate gene in drug discovery ([152]).

Nonetheless, representing the genome of a sample as variants on the reference is an approximation of the genome. In other words, there are cases where the SNPs and SVs are insufficient to model the genome. One example is the major histocompatibility complex (MHC) region in human genomes, which plays an important role in medical applications. We need diploid *de novo* assemblies of each sample to investigate these genomic sequences thoroughly.

In this regard, one approach to obtain a diploid assembly is to use a reference genome, find heterozygous SNVs and SVs, phase these variants, and determine large phased blocks ([40]). `DeepVariant` ([118]), `longshot` ([36]), and `cuteSV` ([56]) are popular software to call variants, and `LongPhase` ([77]) is the latest software that phases SVs and SNVs. As long as we can align the reads accurately, variant calling and phasing methods are robust to sequencing errors, and this approach correctly phases most variants found in a reference. As a result, it has been a *de facto* standard for years ([40]).

However, this approach fails when the alignments of the reads are unreliable, and thus the variant calling is challenging, especially in the following two cases. One case is when the genome of a sample contains large segmental duplications (SDs) that are not represented in the reference, and alignments mix up these SDs (Fig. 2.1a, center). The other case is when two haplotypes contain large SVs compared to the reference (Fig. 2.1a, right).

An alternative approach is to use PacBio's HiFi reads. Their high accuracy (> 99.9%) enables us to accurately reconstruct diploid contigs directly by tightening the identity threshold of the overlaps between reads or by building a *de Brujin* graph with long k-mer matches($k > 1000$). `Hi-Canu` ([105]), `HiFiASM` ([22, 23]), and `Verkko` ([122]) are software in this group. Since this approach does not depend on reference sequences, it is free from the issues in the phasing-based approach.

Nonetheless, to distinguish haplotypes in highly homozygous regions longer than HiFi reads (up to 30 Kbp), this approach needs other data for linking regions that are >100 Kb apart. These data include Hi-C, Illumina's trio data, strand-seq, or ONT's ultra-long reads ([122, 23]). For example, currently, `Verkko` recommends using 50-fold HiFi reads, 50-fold ONT reads longer than 100 Kbp, and 50x of parental Illumina reads or Hi-C reads to achieve a nearly complete diploid assembly. These requirements are costly and can be a budget bottleneck when we try to represent human genetic diversity by assembling diploid genomes on a population scale.

To complement the phasing-based approach, my proposed software, `JTK`, produces the diploid assembly of the hard-to-assemble region in the genome, such as MHC and KIR, with a single sequencing technique (60-fold ONT reads). It takes three steps to assemble nearly identical regions and large SVs simultaneously. (1): `JTK` samples kilobase-scale subsequences (*chunks*) from the reads and aligns these chunks to the reads. By using a relaxed similarity threshold, a chunk can represent similar sequences, such as SDs in the target region, which I call *copies* of the chunk. By aggregating similar sequences into the same chunk, I can represent the target region compactly. (2): `JTK` finds SNVs on these chunks and separates the chunks into each copy. To exhaustively enumerate SNVs on a chunk, `JTK` introduces each possible SNV to the chunk and accepts it as an actual SNV if the alignment scores of many reads increase. (3): `JTK` determines the order of these separated copies in the target region by using the reads and constructs a graph based on the orderings of these copied in the genome. Then, it produces the assembly by traversing the graph.

On the short synthetic haplotypes, `JTK` could estimate haplotypes even when there was only one SNV, suggesting that `JTK` could separate a chunk into copies very sensitively and accurately. On the synthetic haplotypes and 60-fold erroneous long reads, `JTK` outperformed other ONT-based assemblers such as `PECAT` ([100]) or `Phasebook` ([84]) both contiguity and accuracy.

These results showed `JTK` could produce megabase-scale diploid genomes from 60-fold erroneous long reads. In the following two chapters, I will use `JTK` on the real mitochondrial and human datasets to elucidate the diversity of these regions.

2.2 Method

2.2.1 Overview

As long reads contain 5% to 15% errors, it is hard to construct the diploid assembly of the target region using traditional approaches. Also, since two haplotypes in the target region contain significant variants, such as SDs and SVs, it is challenging to construct a haploid assembly first by squishing the variants between haplotypes.

To avoid these issues, JTK first samples subsequences (*chunks*) from the reads (Fig. 2.1b). Each chunk represents similar sequences in the underlying target region, such as repeats with high similarity and homologous sequences, which I call *copies* of the chunk. By selecting chunks so that they cover the target region uniformly, I represent the target region by these chunks. Then, JTK decomposes each chunk into its copies, determines the order of these copies in the target region by the reads, and constructs a partially phased assembly graph. Finally, it produces a fully phased assembly by leveraging the long reads (Fig. 2.1b).

While traditional assemblers rebuild the genome from overlaps between reads, JTK assembles the genome gradually from chunks. Using these chunks, JTK can capture heterozygous insertions and deletions by the presence or absence of chunks from these SVs in the reads. JTK also captures SNVs between haplotypes by finding SNVs on the chunks. In addition, it no longer needs read-vs-read alignments, which can be unreliable.

2.2.2 Determining the chunks from long reads

The initial step of JTK is to determine a set of 2000 bp strings (*chunks*) to represent the underlying target region. Since we do not know the sequence of the target region but know the approximate length of the region, G bp, I randomly sampled L bp substrings in G/L times from the reads as chunks so that I have one-fold coverage by the chunks over the target region. I will discuss how to determine L in Section 2.2.3, but shortly, I use $L = 2000$ by default.

Ideally, these chunks do not overlap and are uniformly distributed on the target region. However, the target region can have sub-regions without sampled chunks, and some chunks overlap. The sub-regions without sampled chunks make gaps in the assembly, and I need to fill these gaps. Also, overlapping chunks are redundant and thus slow down the downstream analysis.

To fill the gaps in the target region, I map the chunks to the long reads, search substrings longer than 2000bp in reads where any chunks do not align, and sample new chunks from these substrings. I repeat this additional sampling until I can not get any additional chunks.

I overlap chunks with each other by `minimap2 --eqx -P` and iteratively remove chunks with the highest number of overlaps until I have no overlaps.

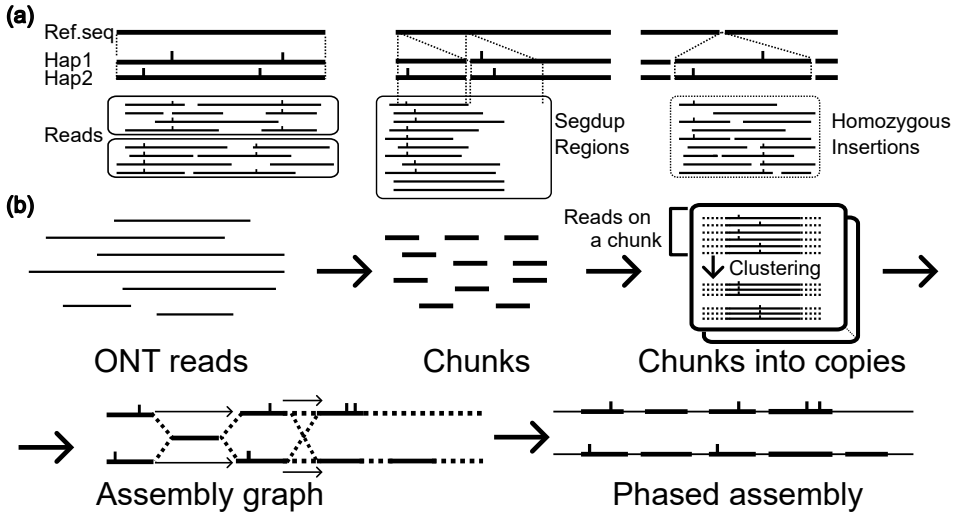

Figure 2.1: (a): The issues of the reference-based phasing. The short ticks are the variants on the reference or th reads. Left: when the reference and two haplotypes are well correlated, I can map reads, find variants, and phase these reads. Center: When the reference underrepresents SDs, the alignments mix up these SDs, and phasing becomes inaccurate. Right: When the reference does not represent the SVs in the genome, I can not align reads, and these reads remain unmapped and unphased. (b): The overview of `JTK`. It starts with long reads. Then, it randomly samples chunks from the reads. After mapping these chunks to the reads, `JTK` separates them into their copies. Then, it constructs a partially phased assembly graph as in the left-bottom figure and resolves the remaining regions to get a fully phased assembly.

After these two follow-ups, the sampled chunks do not overlap, and there is no under-sampled region in the underlying region.

I then align these chunks to the reads and remove errors in the chunks using pair hidden Markov models (pHMMs). I fit the parameters of the pHMMs based on these alignments. To get accurate chunks, I repeat the process of aligning reads to chunks, fitting the parameters of the pHMMs, and correcting errors in the chunks several times (see Section 2.3.3 for a detailed explanation).

After correcting errors in the chunks, I re-align the reads to the chunks, allowing errors of up to 20%, so that regions that are nearly identical to a chunk are represented as the same chunk. I refer to these regions as *copies* of the chunk, and the number of these regions as the *copy number* of the chunk. I will split the chunks into their copies to create a diploid assembly. However, before this step, it is important to accurately estimate the copy numbers of the chunks.

2.2.3 Estimating the copy numbers of chunks

I can approximate these numbers by dividing the number of alignments between the reads and the chunks by the genome-wide haploid coverage. For example, if the haploid coverage is 20 and there are 80 alignments between a chunk and the reads, I estimate the copy number of it as four (= 80/20). To make this approximation accurate, I consider the copy numbers of other neighboring chunks. For example, as shown in the second graph in Fig. 2.11a, if a chunk is adjacent to two chunks with copy numbers equal to two, I impose a constraint that the copy number of that chunk should be four. I devise an optimization problem to consider this constraint and the coverage. See Section 2.3.2 for a detailed explanation.

To determine the length of the chunk, L, I consider the copy numbers and the coverages of the chunks. Specifically, longer chunks would be less repetitive because the chance to include the unique flanking region increases. Since breaking down a chunk into its copies becomes more challenging as the copy number grows, it is preferable to have longer and less repetitive chunks. In contrast, shorter chunks have more reads to be aligned. Because I will use variants on a chunk to separate the chunk into copies, and these variants are easy to detect if the coverage is high, it is better to have shorter chunks. After testing several values of L, I found that $L = 2000$ balances the above tradeoff in the current long reads.

In summary, I have the chunks, the alignments between these chunks and the reads, and the copy numbers of these chunks. As a chunk can represent several regions with high similarity, I need to separate the chunk into multiple copies to produce a diploid assembly, which I will carry out using the reads and the alignments.

2.2.4 Separating the chunks into copies

Here, I briefly describe how JTK decomposes a chunk into its copies. The detailed method (Section 2.3.3 and Section 2.3.5) describes the implementation details and rationales behind JTK's approach.

Given the estimated copy number K of a chunk, I separate the chunk into K copies based on the variants found on the chunk. For example, suppose the chunk is in an SD or, equivalently, in a loop in the assembly graph (the four copy regions in Fig. 2.11a). In that case, I need to separate the chunk into three or more copies, each corresponding to a copy in the target region. Given that I find variants on the chunk by using the reads aligned to the chunk, I can consider separating the chunk into the K copies as separating the reads into the K clusters based on the variants in the reads. Thus, in this section, I describe how to find the variants and separate the reads on the chunk. Traditionally, this problem has been called a phasing or a haplotyping problem in the setting of diploid assembly where the copy number is two ($K = 2$). Nonetheless, I use the term *separation* and *clustering* to highlight that the copy number is not two in general.

To find variants on a chunk, I exploit the idea proposed in the LongShot ([36]), which models the error pattern of a sequencer. Shortly, it can compute the probability of generating a read from a chunk, which I call the *likelihood* of the read.

Specifically, suppose there are N reads, $R = \{r_1, \cdots, r_N\}$, on a chunk c. I propose a two-step approach to call variants and separate the reads R into K clusters, $R_1, \cdots, R_K$.

First, to exhaustively enumerate SNVs on a chunk, JTK introduces each possible SNV to the chunk and records how the log-likelihood changes for each read r. Here, I specify an SNV by its edit operation e (substitutions, insertions, or deletion) and its position i on the chunk. Then, I can denote the table of these changes as $P[r][i][e]$, which I call a *perturbation matrix*. I can compute the table in $O(NL^2)$ time, where L is the maximum length of the reads. For example, $P[r_1][100][del]$ is how much the log-likelihood of the first read changes by deleting the 100th base of the chunk, and r_1 supports this deletion if $P[r_1][100][del]$ is positive. Thus, if $P[r][100][del] > 0$ holds for around half of the reads, this deletion is likely to be an actual variant between the copies of the chunk.

Precisely, for each possible SNV (i, e), I sum up the positive value of the perturbation matrix over the reads, $\sum_{r \in R} \max(P[r][i][e], 0)$. This value gives how much likelihood I can improve by introducing the SNV on the chunk, and I collect SNVs giving a large value of the sum. I denote these collected SNVs as $U = \{(i, e)\}$.

Note that the collected SNVs (U) can contain false positives, which lowers the accuracy of the following clustering. Thus, I filter out the unreliable variants by applying the four filter functions below.

1. The reads supporting the variants have no bias in the strand directions.

2. The variant is not in a long homopolymer run (default: < 3).

3. The number of supporting reads is significantly large (p-value $< 0.05/L$, I use Bonferroni correction).

4. The maximum gain of the log-likelihood ($\sum_r \max(0, P[r][i][e])$) is significantly large and rejects the null hypothesis of random error (p-value $< 0.05/L$).

Second, to resolve the reads R into the K clusters, I maximize the following objective function with respect to the clustering of the reads, $R_1, \cdots, R_K$, such that the union is R, and R_k and $R_{k'}$ are disjoint for all $k \neq k'$.

$$\sum_{k=1,\cdots,K} \sum_{(i,e)\in U} \max\left(\sum_{r\in R_k} P[r][i][e], 0\right) \tag{2.1}$$

Shortly, $\sum_{r\in R_k} P[r][i][e]$ is how much log-likelihood would increase in the reads in R_k by editing the i-th base of the chunk by e. To ignore the case where this value is negative and the cluster does not support the operation (i, e), I put $\max(\_, 0)$ in (2.1).

This maximization problem is NP-hard because it is NP-complete to check the presence of a clustering with a score more than a given score (a reduction is from MAX CUT). Thus, I use a sampling algorithm to find a nearly optimal solution (Algorithm 2) after obtaining draft clustering by K-means++ ([9]).

Since I have not proved that this algorithm is an MCMC with detailed balanced conditions, it is possible that this algorithm may get stuck in some narrow regions of the solution space, i.e., all possible clusterings. To show that this is not the case, I will evaluate the performance of Algorithm 5 in the Results section.

Algorithm 2 Clustering of reads

Input: A perturbation matrix P, the variants U, and the number of cluster K
Output: $R_1, \cdots, R_K$ (a clustering of $\{r_1, \cdots, r_N\}$)

1: $R_1, \cdots, R_K \leftarrow$ randomly partition $r_1, \cdots, r_N$ into K clusters.
2: $p \leftarrow$ compute the objective function (2.1) ▷ Greater is better.
3: **for** T times **:** ▷ T=2000 × n by default
4: Select R_i and R_j at random and move a random element in R_i to R_j
5: $\hat{p} \leftarrow$ compute the objective function (2.1)
6: **if** $p < \hat{p}$ **:**
7: $p \leftarrow \hat{p}$
8: **else**
9: Set p to $\hat{p}$ with a probability $exp(\hat{p} - p)$; otherwise, restore the latest change made at Step 4
10: **return** $R_1, \cdots, R_K$

Because of the high error rate in the long reads, the clustering (Algorithm 2) may fail to assign a read to the correct cluster. To fix this issue, JTK further

improves the clustering on a chunk by integrating the results of nearby chunks (Section 2.3.6).

In summary, to separate the chunks into their copies, I assign the reads on each chunk into its copies by calling variants by the perturbation matrix followed by the clustering (Algorithm 2). The remaining challenges are determining the order of these separated chunks in the target region and resolving regions without any variants.

2.2.5 Serializing the separated copies

After the clustering, I infer the ordering of copies of the chunks in the target region by constructing a graph where the nodes are the copies of the chunks, and an edge links between two nodes adjacent in a read.

The graph is not necessarily fully phased into the two haplotypes at this stage because the chunks corresponding to completely homozygous regions in the target region do not have any variants(Fig. 2.11a the third graph). I solve these homozygous regions by checking the co-occurrence of the nodes in the same reads and by finding the correct path of each haplotype.

To generate the final assembly, I first concatenate chunks inside the path in the graph to obtain the draft assembly and then take consensus among the reads by using the perturbation matrix on the draft assembly. Specifically, I compute the perturbation matrix on the draft sequence s and take the sum over the reads to get $P_s[i][e] = \sum_r P[r][i][e]$. If there are any pairs of a position and an edit distance (i, e) such that $P_s[i][e]$ is positive, I put them into the draft sequence to get a better consensus.

After taking consensus, I obtain the phased assembly, whose nodes are labeled either haplotype-phased or multi-copy (Fig. 2.11a rightmost graph). I use the haplotype-phased contigs in the downstream analysis.

2.3 Detailed methods

2.3.1 Determining the chunks from long reads

The initial step of `JTK` is to determine a set of 2000 bp strings (*chunks*) to represent the underlying target region. Assuming the approximate length of the region is G bp, I randomly sampled 2000 bp substrings in $G/2000$ times from the reads as chunks. As a result, I have one-fold coverage by these chunks over the target region. Ideally, these chunks do not overlap and are uniformly distributed on the target region. However, the target region can have sub-regions without sampled chunks and make gaps in the assembly. I need to fill these gaps. Also, some chunks overlap, and these redundant chunks slow down the downstream analysis.

To fill the gaps in the target region, I map the chunks to the long reads, search substrings longer than 2000bp in reads where any chunks do not align and sample

new chunks from these substrings. I repeat this additional sampling until I can not get any additional chunks.

I overlap chunks with each other by `minimap2 --eqx -P` and iteratively remove chunks with the highest number of overlaps until I have no overlaps. Note that the problem of removing overlapping chunks has a graph-theoretic formalization.

Definition 2.3.1 (Removing overlapping chunks)**.** Let V be a set of chunks. Then, I represent an overlap between two chunks as a pair of elements in V. In other words, $E \subseteq V \times V$ be a set of overlaps, i.e., $(i, j) \in E$ if and only if the i-th chunk and the j-th chunk overlap. Then, The removing overlapping problem is defined as follows:

- Input: a set of chunks V and a set of overlap $E \in V \times V$
- Output: The minimum set of the chunks V' such that for all $(i, j) \in E$, either i or j is in V'. In other words, when I remove V' from V, there is no edge remaining.

The decision version of this problem is the same as INDEPENDENT-SET, a well-known NP-complete problem, which justifies the heuristics I use in this step.

After these two follow-ups, the sampled chunks do not overlap, and there is usually no under-sampled region in the underlying region.

2.3.2 Estimating the copy numbers of chunks

A chunk sampled in Section 2.3.1 can represent multiple similar sequences in the target region, such as homologous sequences between haplotypes or similar repeats, which I call *copies* of the chunk. To fully assemble the region, I separate a chunk into each copy except homozygous regions without any variants. To this end, I first need to determine how many copies a chunk has in the target region, which I refer to as the *copy number* of the chunk.

Intuitively, to estimate copy numbers, I align the chunks to the reads, which tell two things. For one thing, they tell how many times a chunk is aligned to the reads. I can approximate the copy number by dividing it by the haploid coverage. For the other, the alignments tell the positions of the chunks aligned in the reads, which reveal how the copy numbers relate to each other. For example, if two chunks are always aligned side-by-side in the reads, these chunks must have the same copy number. I can fine-tune the approximated copy numbers from this information.

In the following section, I explain the algorithm to estimate the copy numbers. I first construct a graph called *chunk graph* from the chunks and the reads. Then, I convert the chunk graph into another graph called *chunk double-stranded graph* to derive an efficient algorithm for estimating copy numbers.

Construction of a chunk graph from the chunks, the reads, and the alignment between them.

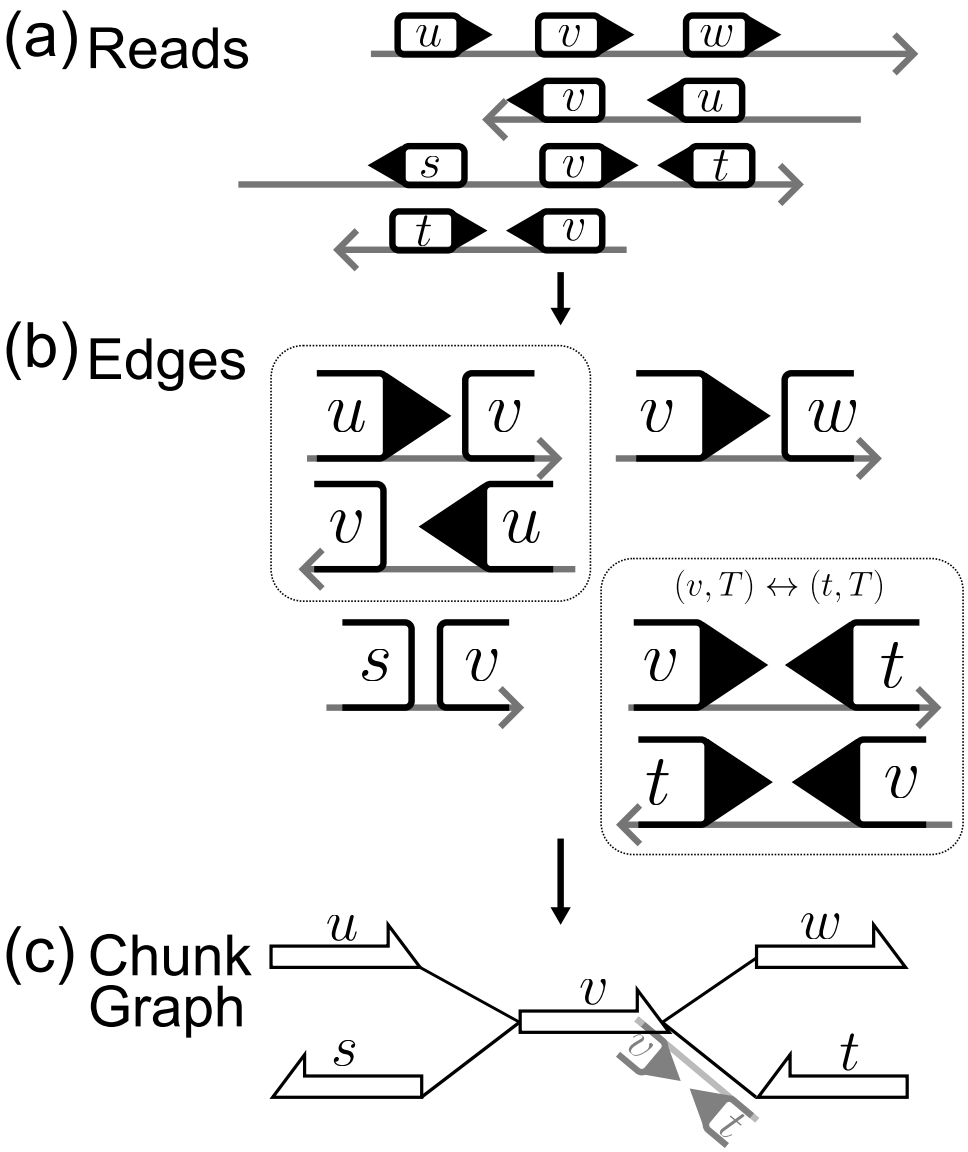

Figure 2.2: The conversion from the reads to the chunk graph. (a): A set of the reads (gray arrows indicate the sequence direction). The arrows at the side of the chunks indicate the alignment direction. (b): The set of edges from the reads. Each edge represents a connection of nodes and the direction of these nodes. I regard the edges in the dotted box as the same edge to erase the strand information of the reads; For example, the third and the fourth read have the same edge, $(v, T) \leftrightarrow (t, T)$. (c): The chunk graph converted from (b). I put the edge $(v, T) \leftrightarrow (t, T)$ in the graph as an example.

Suppose two chunks are mapped to the same read, and there are no other chunks between them in the read. Since the read is a subsequence of the target region, these two chunks should be close to each other in the target region, and I call these two chunks *adjacent*. [1] The alignments between the chunks and the reads tell which pairs of chunks are adjacent. I can summarize these adjacencies among the reads by creating a graph where the nodes are the chunks, and I draw edges between adjacent chunks (Fig. 2.2).

Here, to make the graph represent the underlying target region, I need to consider two factors – the alignment direction and the strand direction.

First, a chunk is aligned either forward or reverse direction. For example, suppose three chunks u, v, and w are aligned to the same read in the forward direction (the first read in Fig. 2.2a). I have two edges in this read; one connects the last base of u to the first base of v, and the other connects the last base of v and the first base of w. These bases are different, and I distinguish them by introducing H and T to represent a chunk's first (*head*) and last (*tail*) bases. For example, I represent the edge connecting the last base of u and the first base of v as an edge from (u, T) to (v, H).

Second, although the reads have two directions (forward or reverse), I regard these directions as the same because they are essentially the same. For example, the first read in Fig. 2.2 has two chunks, u and v, aligned in the forward direction in this order. In contrast, the second read has two chunks, v and u, aligned as the reverse complement. These edges should be the same in the graph because they represent the same situation – they connect the last base of u and the first base of v. Thus, I treat an edge as undirected, i.e., I consider $(u, T) \leftrightarrow (v, H)$ to be the same as $(v, H) \leftrightarrow (u, T)$.

By considering these two factors, I represent the reads (Fig. 2.2a) as a *chunk graph* (Fig. 2.2c) $G = (V, E)$ as follows:

- The nodes V are the chunks.
- The edges $E \subseteq (V \times \{H, T\}) \times (V \times \{H, T\})$ contains $(v, p) \leftrightarrow (u, q)$ if and only if there is a read that connects the p (Head or Tail) position of the chunk v and the q (Head or Tail) position of u.
- Rigorously, I should define $V \times \{H, T\}$ as the nodes, but for simplicity, I refer to the set of chunks V as *nodes*.

A chunk graph has three types of coverages to approximate the copy numbers of the chunks. First, for each node v, I count the number of alignments between the reads and the chunk represented by v and call it the coverage of the node, c_v.

Second, for each edge e, I count the number of e in the reads, i.e., the occurrences of the adjacency, and call it the coverage of the edge, c_e.

[1] Precisely, two copies of these chunks in the underlying target region are adjacent.

Lastly, assuming the target region is in a diploid genome, I divide the average coverage of the chunks by two and denote it as the global haploid coverage, $\bar{c}$. Formally,

$$\bar{c} = \sum_{v \in V} c_v / 2|V| \tag{2.2}$$

Here, I assume that most of the chunks have a copy number equal to two. Also, this parameter $\bar{c}$ can be set externally to use JTK in the situation where we can assume that the data sets are virtually the haploid genomes.

Based on these coverages, I approximate the copy numbers of nodes as $c_v/\bar{c}$ and those of edges as $c_e/\bar{c}$. [2]

I improve these simple estimations by considering the structure of the chunk graph. For example, suppose the node v in Fig. 2.2 corresponds to a segmental duplication in the underlying diploid genome. Also, suppose that the nodes u, s, w, and t are flanking regions to the segmental duplication, and the copy numbers of these nodes are two. Here, the copy number of v should be four, which is the same as the sum of the copy numbers of u and s, or that of w and t.

In general, a node's copy number is equal to the sum of the copy numbers of the edges at either end (the dotted boxes in the chunk graph in Fig. 2.3a). In the next section, I present the optimization problem considering these consistency conditions and the coverages (c_v, c_e, and $\bar{c}$).

Converting a chunk graph to a double-stranded graph to estimate copy numbers

Formally speaking, I denote the copy number of a node v and an edge e as $\#_N(v) \in \mathbb{N}$ and $\#_E(e) \in \mathbb{N}$, respectively. Because these copy numbers should explain the observed coverages, the estimated copy numbers would minimize the sum of squared error (SSE):

$$\mathrm{SSE} = \sum_{v:\mathrm{nodes}} \|c_v - \#_N(v)\bar{c}\|^2 + \sum_{e:\mathrm{edges}} \|c_e - \#_E(e)\bar{c}\|^2 \tag{2.3}$$

Also, as noted in the previous section, the consistency condition of the copy numbers below needs to be satisfied for each node v:

$$\#_N(v) = \sum_{e=(\_,\_)\leftrightarrow(v,H)} \#_E(e) \quad \text{if there is an edge to } (v,H) \tag{2.4}$$

$$\#_N(v) = \sum_{e=(v,T)\leftrightarrow(\_,\_)} \#_E(e) \quad \text{if there is an edge from } (v,T) \tag{2.5}$$

Combining the constraints on the graph, I have the following constrained minimization problem:

[2] Precisely, I round them to the nearest integer.

- Input: a chunk graph $G = (V, E)$ and coverages c_v, c_e, and $\bar{c}$.
- Output: Copy numbers on the nodes ($\#_N(v) \in \mathbb{N}$) and edges ($\#_E(e) \in \mathbb{N}$), which
  - minimize the sum of squared error (Eq. (2.3)).
  - satisfy the consistency conditions (Eq. (2.4), Eq. (2.5)) for each v.

How can we solve this optimization problem? One approach is to change $\#_N(v)$ and $\#_E(e)$ to reduce the SSE while maintaining consistency. As $\#_N(v) = 0$ and $\#_E(e) = 0$ for all nodes and edges satisfy the consistencies, I initialize the copy numbers as zero. The problem is how to change these copy numbers without breaking the consistencies.

Some paths and cycles in the chunk graph are helpful for this purpose. For example, think about the path in Fig. 2.3a consisting of u, $(u, T) \leftrightarrow (v, H)$, v, $(v, T) \leftrightarrow (t, T)$, and t. Incrementing all the copy numbers of nodes and edges in this path does not break the consistency conditions because whenever I increment the right side of condition (2.4) or (2.5), I increment the left side of these conditions and vice versa. I call these paths and cycles *balancing* because they "balance" the equation (2.4 and 2.5).

I summarize my approach so far as follows:

1. I initialize the copy numbers of edges and nodes to be zero.
2. I iteratively update the copy numbers along a balancing path or cycle to reduce the SSE (Eq. (2.3)).

Nonetheless, not all paths are balancing paths. For example, u, $(u, T) \leftrightarrow (v, H)$, $(v, H) \leftrightarrow (s, H)$, s is a valid path in Fig. 2.3a, but I can not increment the copy numbers along this path without breaking the consistencies. How can we modify the structure of the chunk graph so that we can find balancing paths and cycles easily?

To this end, I introduce the strand information to the chunk and convert the chunk graph G into another directed graph as follows (Fig. 2.3b). Intuitively, after this conversion, every path should interchangeably traverse an edge and a node in the chunk graph.

1. Substitute v with four nodes representing the first/last base of the nodes and the forward/reverse strand. Formally, let H and T be the head (H) and the tail (T) of the nodes as before, and F and R be the forward strand (F) and the reverse strand (R). Then, I substitute v with (v, H, F), (v, T, F), (v, H, R), and (v, T, R). For example, (v, H, F) is the head position in the forward strand of v.
2. For each v, I connect the head node to the tail node in both strands. Namely, I add two edges, $(v, H, F) \to (v, T, F)$ and $(v, H, R) \to (v, T, R)$. Hereafter,

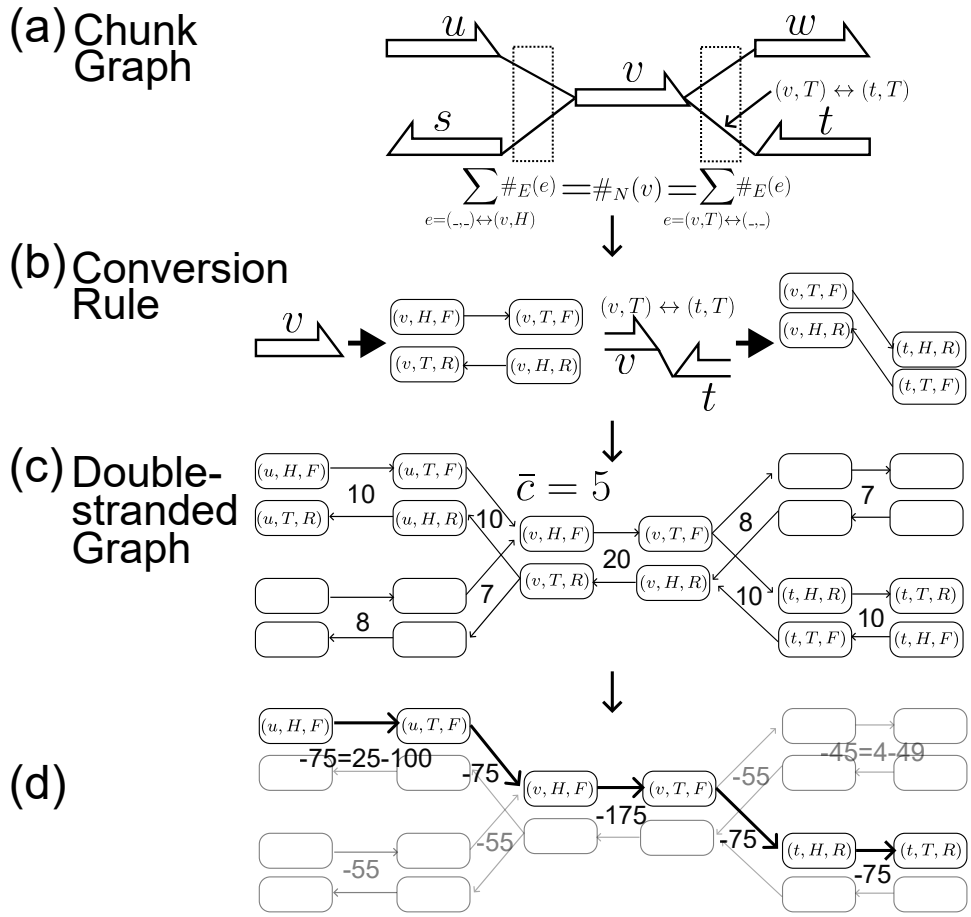

Figure 2.3: (a): The consistency condition on the chunk graph (Fig. 2.2c). Each head or tail position of the nodes has a consistency condition if there are edges connected to it. (b): The conversion rule for the double-stranded graph. I replace each node v with four nodes and two edges. I also replace each edge e with two edges. (c): The double-stranded graph converted from (a) by the conversion (b). The average coverage $\bar{c}$ is 5, and I label each edge by the coverages. (d): Initial iteration of the algorithm to estimate the copy numbers. I label each edge by how the sum of squared error decreases. For example, because I initialize all copy numbers as zero, the average coverage $\bar{c}$ is five, and the coverage of the edge $(u, H, F) \longrightarrow (u, T, F)$ is ten, the weight of this edge is $\|10 - 1 \times \bar{c}\|^2 - \|10 - 0 \times \bar{c}\|^2 = -75$. The thick edges show the path selected in this iteration. (u, H, F) is the source and (t, T, R) is the sink of this path.

I will not add any edges from the heads, i.e., $(\_, H, F)$ or $(\_, H, R)$. Thus, whenever I reach $(\_, H, F)$ or $(\_, H, R)$, I need to go to the corresponding tail positions, i.e., $(\_, T, F)$ or $(\_, T, R)$. This property forces us to traverse an edge and a node interchangeably in the original chunk graph.

3. For each edge in the original graph, I split the edge into two directed edges representing the forward and reverse directions. For example, the edge $(v, T) \leftrightarrow (t, T)$ would be $(v, T, F) \rightarrow (t, H, R)$ and $(t, T, F) \rightarrow (v, H, R)$ (Fig. 2.3b). In general, I replace an edge $e = (v, p) \leftrightarrow (u, q)$ as follows. The first edge is a directed edge from v to u.

$$
\begin{aligned}
(v, T, F) &\rightarrow (u, H, F) \text{ if } p = T \text{ and } q = H \\
(v, T, F) &\rightarrow (u, H, R) \text{ if } p = T \text{ and } q = T \\
(v, T, R) &\rightarrow (u, H, F) \text{ if } p = H \text{ and } q = H \\
(v, T, R) &\rightarrow (u, H, R) \text{ if } p = H \text{ and } q = T
\end{aligned}
$$

As the edge e is undirected, the second edge is a directed edge from u to v defined similarly.

$$
\begin{aligned}
(u, T, F) &\rightarrow (v, H, F) \text{ if } q = T \text{ and } p = H \\
(u, T, F) &\rightarrow (v, H, R) \text{ if } q = T \text{ and } p = T \\
(u, T, R) &\rightarrow (v, H, F) \text{ if } q = H \text{ and } p = H \\
(u, T, R) &\rightarrow (v, H, R) \text{ if } q = H \text{ and } p = T
\end{aligned}
$$

I call this graph the *double-stranded graph* because I introduce F and R to express the double-strandedness. I also refer to a node without any incoming edges as a *source* node and a node without any outgoing edge as a *sink* node.

The double-stranded graph has a useful property; all cycles and all paths from source nodes to sink nodes correspond to balancing cycles and paths in the original chunk graph. For example, the thick nodes and edges in Fig. 2.3c is a path from the source node (u, H, F) to the sink node (t, T, R) corresponding to the balancing path consisting of u, $(u, T) \leftrightarrow (v, H)$, v, $(v, T) \leftrightarrow (t, T)$, and T in the original chunk graph. By definition of the balancing path, changing copy numbers along with this path does not break the consistency condition.

Thus, to optimize the SSE (Eq. (2.3)), I initialize the copy number of the nodes ($\#_N(v)$) and edges ($\#_E(e)$) as zero and change copy numbers along a cycle or a path in the double-stranded graph until I can not decrease the SSE (2.3). For example, suppose we set all the copy numbers to zero and then increment the copy numbers along the path depicted in Fig. 2.3d. The squared error of the edge $(u, H, F) \rightarrow (u, T, F)$ changes from $\|10 - 0 \times 5\|^2 = 100$ to $\|10 - 1 \times 5\|^2 = 25$, decreasing by 75. Summing up all the changes, the SSE decreases by 475. I continue this procedure until the SSE does not decrease.

Nonetheless, the above procedure is not rigorously defined in two points.

First, I need to determine how much the copy numbers I will change along a path or cycle in each iteration. In addition, I need to select a path or cycle from possibly many paths and cycles.

Shortly, for the first issue, I decide whether to increment or decrement the copy numbers at each iteration at random. As the copy numbers are integers, it is reasonable to assume that I can reach a nearly-optimal solution by incrementing or decrementing the copy numbers.

For the second issue, for each node and edge, I compute how the SSE (Eq. (2.3)) will change if I change the copy number of the node or edge. By summing up these changes along a path or cycle, I can also estimate how the SSE will change in total and thus prioritize the paths and the cycles.

Here, suppose we are going to increment the copy numbers. Then, for each edge in the double-stranded graph, I compute how the SSE changes when I increment the copy number of it. Because the increment is the minimum unit of the change in integers, I call the change *gradient* of the edge.

The two edges connecting the head node and the tail node of $v \in V$ in the original graph share the same gradient. Precisely, the gradient of the $(v, H, F) \to (v, T, F)$ and $(v, H, R) \to (v, T, R)$ is as follows:

$$\|c_v - (\#_N(v) + 1)\bar{c}\|^2 - \|c_v - \#_N(v)\bar{c}\|^2 \tag{2.6}$$

Also, I have converted an edge $e \in E$ into two edges, and they share the same gradient below:

$$\|c_e - (\#_E(e) + 1)\bar{c}\|^2 - \|c_e - \#_E(e)\bar{c}\|^2. \tag{2.7}$$

Suppose there is a cycle or a path from a source to a sink node in the double-stranded graph (Fig. 2.3d shows the latter case), and the sum of the gradient in it is negative. Then, if incrementing the copy numbers along this cycle or path indeed decreases the SSE (Eq. (2.3), I increment these copy numbers.

Note that there are corner cases in this algorithm. For example, if I select a cycle containing both $(v, H, F) \to (v, T, F)$ and $(v, H, R) \to (v, T, R)$, I traverse the v in the original chunk graph twice. In this case, the sum of the gradient in the cycle is not equal to the change of the SSE on the incrementation along the cycle. Thus, it is unclear whether SSE would decrease by incrementing the copy numbers along this cycle.

I can avoid these cases with two heuristics. First, I confirm whether the SSE decreases before incrementing copy numbers to confirm that the SSE monotonously decreases during iterations. Second, if I can not confirm that the SSE does not decrease, I search for another path or cycle that does not contain these corner cases. Although there is no proof that I can find it, this search is usually successful in real datasets.

In the case of decrementing copy numbers, I can have almost the same algorithm with obvious modifications on the gradients, i.e., (2.6) and (2.7).

To find negative cycles or paths, although there is a dedicated algorithm ([157]), I employ the $O(|V||E|)$-time Bellman-Ford algorithm. Here, $|V|$ and $|E|$ are the number of nodes and edges, respectively. As the number of nodes is usually around thousands and the number of edges is almost linear to that of nodes, the Bellman-Ford algorithm is feasible in practice. Also, as I use 2000 bp long chunks by default, the copy numbers are usually less than 15, requiring at most 15 increments of the copy numbers along cycles or paths to reach an accurate estimation.

In summary, I employ the following four-step approach to accurately estimate the copy numbers of the chunks.

1. Align the chunks to reads (Fig. 2.2a).
2. Construct the chunk graph from the alignments (Fig. 2.2b).
3. Convert the chunk graph into the double-stranded graph (Fig. 2.3c).
4. Iteratively refine the estimated copy numbers by incrementing or decrementing the copy numbers along a path or cycle (Fig. 2.3d)

This four-step procedure considers not only the occurrences of the chunks in the reads but also the connections of the chunks on the target region to estimate the copy numbers accurately.

Note that this is not the only way to compute the number of copies of a chunk. For example, there may be other ways to optimize the SSE under the constraints, such as Gibbs sampling.

Also, there is a case where the proposed algorithm may not work correctly. For example, let C_1, C_2, C_3 be DNA sequences. If the genome is the concatenation of $C_1C_1C_2C_3$ and C_3 in this order, there would be no sink or source nodes in the chunk graph derived from this genome. Therefore, I cannot update the copy number, so the estimated copy number across all chunks is zero. However, in this work, I have chosen the target regions so that they do not have segmental duplications at either end to avoid this corner case.

The following section and Section 2.2.4 explain how to use this information in the clustering step.

2.3.3 Consensus and variant calling by a pair-hidden Markov model

(In this section, let an alphabet Σ be $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$. Also, a *string* s, a *read* r, and a *chunk* c are elements in Σ^*.)

2.3.4 Notes on the strandedness of the reads

Before I get into the details, let me quickly note how I handle the strand information in this section. First of all, we cannot determine strand information without

special techniques such as strand-seq ([37]). So when I handle subsequences of reads aligned to a chunk, I take the reverse complements of the reads aligned in the reverse direction. Although it is appropriate to remember whether I take the reverse complement of these reads in downstream analysis, the current implementation discards them to keep the method simple.

A pair-hidden Markov model and basic algorithms

Consensus, i.e., polishing draft contigs, is one of the most crucial steps in the assembly as consensus errors directly affect the downstream analysis. Specifically, given that the average difference between haplotypes is around 0.1%, we need more than 99.9% accuracy to reliably infer the haplotype divergence.

Also, variant calling should be very accurate when I separate chunks into their copies which can be as similar as 99.9% (see Section 2.2.4). This task is not trivial because the high error rates in the reads make it challenging to find true variants between haplotypes or segmental duplication.

Thus, I need sophisticated models and algorithms to take consensus and call variants. To this end, I use a pair-hidden Markov model (pHMM) and algorithms on it.

In this subsection, I define a pHMM and explain three algorithms that simulate how a DNA sequencer outputs a read from a chunk and compute the probability of observing a read. These algorithms are building blocks in the consensus and the variant calling.

A pair-hidden Markov model models a DNA sequencer. It has two major parameters modeling the state of a sequencer and the base we observe at each state.

Precisely, a pHMM has three states, M, D, and I, corresponding to the match, deletion, and insertion state to model the consecutive insertion and deletion errors in a read. For example, the probability of observing a deletion error is smaller after a (mis-)match than after another deletion error. In other words, observing consecutive deletion errors is more likely than observing a deletion after a (mis-)match. To model these tendencies, a pHMM has $T_{s,s'}$ representing the transition probability from the state s to s':

$$T = \begin{pmatrix} T_{M,M} & T_{M,D} & T_{M,I} \\ T_{D,M} & T_{D,D} & T_{D,I} \\ T_{I,M} & T_{I,D} & T_{I,I} \end{pmatrix} \tag{2.8}$$

Now, a higher tendency of consecutive deletions can be modeled by setting $T_{M,D} < T_{D,D}$. Of course, the actual error patterns in the long reads are context dependent and more complex than the three-state model. Nonetheless, the more complex a model is, the more elaborate algorithm have to be developed for the model. I assume that the three-state pHMM solves the trade-off between these two complexities. [3]

[3]Here, I use the word "complex" intuitively. It may have something to do with "computational

I determine these transition probabilities based on alignments between chunks and reads. Shortly, I represent an alignment as an array of matches or mismatches (M), deletions (D), and insertions (I), and let an alignment between a read and a chunk be $L = L_0 \cdots, L_{N-1}$, where N is the length of the alignment, and $L_n \in \{\mathrm{M}, \mathrm{D}, \mathrm{I}\}$. Then, to estimate $T_{M,M}$, I divide the number of consecutive matches by the length of the alignments. Precisely[4],

$$T_{M,M} = \frac{|\{n \mid 0 \leq n < N-1, L_n = \mathrm{M}, L_{n+1} = \mathrm{M}\}|}{|\{n \mid 0 \leq n < N-1, L_n = \mathrm{M}\}|}$$

It is easy to generalize the definition to more than two alignments and other values of T.

Each state has observation probabilities for generating a read from a chunk. The match state simulates the substitution errors of a long-read sequencer and outputs a base $x \in \Sigma$ depending on a base $y \in \Sigma$ of the chunk, and I denote this probability as $P_M(x|y)$.

Similarly, the insertion state simulates the base that a sequencer inserts in the reads, and I denote the probability of observing a base $x \in \Sigma$ at the insertion state as $P_I(x)$.

The deletion state always outputs the gap symbol (–) or, equivalently, does not output anything.

I estimate these observation probabilities from alignments between reads and chunks. For example, suppose we have an alignment between a chunk and a read. To estimate $P_M(\mathtt{A}|\mathtt{A})$, I first count the number of matches between a base in the query and the adenine (A) in the chunk. Specifically, let $\mathrm{Aln}(x|y)$ be the number of (mis-)matches in the alignment between the base x in the read and the base y in the chunk. Then,

$$P_M(\mathtt{A}|\mathtt{A}) = \frac{\mathrm{Aln}(\mathtt{A}|\mathtt{A})}{\sum_{x=\mathtt{A},\mathtt{C},\mathtt{G},\mathtt{T}} \mathrm{Aln}(x|\mathtt{A})}$$

Similarly, I estimate $P_I(x)$ by normalizing the count of the base x in the read at the insertion state.

A pHMM models a DNA sequencer, and I use it to generate a read from a chunk (Algorithm 3). Shortly, a pHMM changes its state and moves the position on the chunk that the sequencer reads during sequencing, which I call the *head* on the chunk. Initially, the state is on the match state, and the head is on the first base of the chunk. Then, the state moves among three states, and the head proceeds by one base every time we observe a base at the match or deletion state. Algorithm 3 iteratively makes transitions and observations until the head on the chunk leaves the last base.

I can use this simulation algorithm to evaluate a variant on a chunk such as SNP. Roughly speaking, I simulate long-reads from the chunk to compute how

complexity" or other complexities, but, roughly speaking, I am talking about how it is easy to understand.

[4] I assume the alignment is longer than two, and there is at least one match.

many reads happen to support the variant due to sequencing errors, which enables us to quantitatively assess the variant. I will argue this procedure more deeply in Section 2.3.5.

I denote the probability, or the *likelihood*, that the above algorithm (Algorithm 3) generates a read r from a chunk c by a pHMM H as $L(r|c, H)$. Hereafter, I omit H from the notation when it is obvious from the context. I also note that it is usually defined as $L(c|r)$ to emphasize that the variable is the sequence of the chunk. However, I will use the inverted notation ($L(r|c)$) because it makes the notation of the algorithm easier.

I can compute $L(r|c)$ by the forward algorithm (Algorithm 4) in $O(|r||c|)$ time. Shortly, this algorithm computes the probability that a pHMM outputs a state $s \in \{M, I, D\}$ and a read's prefix from a chunk's prefix. I denote this probability as $L(r_{\dots j}, s \mid c_{\dots i})$. Here, $r_{\dots j}$ is the prefix of r up to the $j-1$-th base, and $c_{\dots i}$ is the prefix of c up to the $i-1$-th base. Also, as the state s is before the condition (|) in $L(r_{\dots j}, s \mid c_{\dots i})$, I can obtain $L(r|c)$ by summing up s over $L(r, s \mid c)$.

Another algorithm computes the probability that a pHMM generates a read's suffix from a chunk's suffix and a state $s \in \{M, I, D\}$, i.e., $L(r_{j\dots}|c_{i\dots}, s)$, which is called the backward algorithm (Algorithm 5). Note that the state s in the backward algorithms is in the condition in $L(r_{j\dots}|c_{i\dots}, s)$. Given the initial state is the match state M, $L(r|c)$ is the same as $L(r|c, M)$.

In the next section, I will explain an algorithm to compute the likelihood of r when introducing an edit operation to c by combining these two algorithms, which has been given by Chin ([24]).

Algorithm 3 Generating a read from a pair-HMM

Input: a pHMM and a chunk c
Output: A read r

$s \leftarrow M$ ▷ Start with the match state
$i \leftarrow 0$ ▷ The head on the chunk c
$r \leftarrow$ the empty string
while $i < |c|$ **:** ▷ Until the pointer leaves the last base
    $s \leftarrow$ select the next state from the probability $T_{s,M}$, $T_{s,I}$, and $T_{s,D}$
    if $s = M$ **:**
        Generate a base x with the probability $P_M(x|c_i)$ and push it to r ▷ c_i is the i-th base of c.
        $i \leftarrow i + 1$
    else if $s = I$ **:** ▷ If the state is the insertion state.
        Generate a base x with the probability $P_I(x)$ and push it to r
    else
        $i \leftarrow i + 1$ ▷ A deletion state does not output base.
return r

Algorithm 4 The forward algorithm for a pair-HMM

Input: a pHMM, a read r, and a chunk c

Output: Likelihoods of outputting a state s and a read's prefix $r_{\cdots j}$ from a chunk's prefix $c_{\cdots i}$, $F_s[i][j] = L(r_{\cdots j}, s|c_{\cdots i})$

Initialize three $(|c|+1) \times (|r|+1)$ array F_M, F_D, F_I with 0
$F_M[0][0] \leftarrow 1$ ▷ The match state is the initial state.
for $i = 1, \cdots, |c|+1$ **:**
    $F_D[i][0] \leftarrow \sum_{s\in\{M,D,I\}} F_s[i-1][0]T_{s,D}$ ▷ $\sum_s$ runs over $\{M, D, I\}$ in other lines.
for $j = 1, \cdots, |r|+1$ **:**
    $F_I[0][j] \leftarrow \sum_s F_s[0][j-1]T_{s,I}P_I(r_{j-1})$ ▷ r_{j-1} is the $j-1$-th base of r.
for $i = 1, \cdots, |c|+1$, $j = 1, \cdots, |r|+1$ **:**
    $F_M[i][j] \leftarrow \sum_s F_s[i-1][j-1]T_{s,M}P_M(r_{j-1}|c_{i-1})$
    $F_D[i][j] \leftarrow \sum_s F_s[i-1][j]T_{s,D}$
    $F_I[i][j] \leftarrow \sum_s F_s[i][j-1]T_{s,I}P_I(r_{j-1})$
return F_M, F_I, F_D

Algorithm 5 The backward algorithm for a pair-HMM

Input: a pHMM, a read r, and a chunk c

Output: Likelihoods of outputting a read's suffix $r_{j\cdots}$ from a chunk's suffix $c_{i\cdots}$ and a state s, $B_s[i][j] = L(r_{j\cdots}|c_{i\cdots}, s)$

Initialize three $(|c|+1) \times (|r|+1)$ array B_M, B_D, B_I with 0
$B_{s\in\{M,D,I\}}[|c|+1][|r|+1] \leftarrow 1$ ▷ $\sum_s$ runs over $\{M, D, I\}$ in other lines.
for $i = |c|, \cdots, 0$ **:**
    $B_D[i][|r|+1] \leftarrow T_{D,D}B_D[i+1][|r|+1]$
    $B_I[i][|r|+1] \leftarrow T_{I,D}B_D[i+1][|r|+1]$
    $B_M[i][|r|+1] \leftarrow T_{M,D}B_D[i+1][|r|+1]$
for $j = |r|, \cdots, 0$ **:**
    $B_D[|c|+1][j] \leftarrow T_{D,I}P_I(r_j)B_I[|c|+1][j+1]$ ▷ r_j is the j-th base of r.
    $B_I[|c|+1][j] \leftarrow T_{I,I}P_I(r_j)B_I[|c|+1][j+1]$
    $B_M[|c|+1][j] \leftarrow T_{M,I}P_I(r_j)B_I[|c|+1][j+1]$
for $i = |c|, \cdots, 0$, $j = |r|, \cdots, 0$, $s = M, D, I$ **:**
    $B_s[i][j] \leftarrow T_{s,M}P_M(r_j|c_i)B_M[i+1][j+1]+$ ▷ from s to the match state.
    $T_{s,D}B_D[i+1][j]+$ ▷ from s to the deletion state.
    $T_{s,I}P_I(r_j)B_I[i][j+1]$ ▷ from s to the insertion state.
return B_M, B_I, B_D

Consensus and variant calling algorithm

I sample non-overlapping kilobase-scale subsequences from the reads and call them *chunks*. However, as long reads are error-prone, these chunks contain errors up to approximately 10%. I need to remove these errors to get an assembly with high accuracy. [5]

To formalize this problem, recall a pair-hidden Markov model simulates a sequencer and computes the likelihood of generating a read r from a chunk c as $L(r|c)$. Maximizing the likelihood is expected to minimize the number of errors in the chunk.

Formally, for a set of reads R and a draft chunk c, I modify c to maximize the log-likelihood defined as (2.9).

$$\sum_{r \in R} \ln L(r|c) \tag{2.9}$$

Although there is no guarantee that I can reach the global maximum of (2.9), the algorithm I will explain in this section achieves more than 99.9% accuracy for 30-fold reads.

Calling variants is similar to taking consensus. Suppose a chunk has an SNV position where the base in the maternal haplotype is `A` and that in the paternal is `C`. Then, if we change the base in the chunk at the SNV location from `C` to `A`, the likelihood of the reads from the maternal haplotype would increase because it would change a mismatch to a match at the SNV. Similarly, by changing the base from `A` to `C`, the likelihood of the reads from the paternal haplotype would increase. Conversely, if we find an edit operation that improves the likelihood on some, but not all, reads, it might represent a variation between haplotypes.

Overall, this section will compute how the likelihood of a read r changes if we apply an edit operation e to a chunk c, denoted as $P[r][i][e]$. In the rest of this section, I show how to compute $P[r][i][e]$ efficiently.

For simplicity, suppose we insert a new base `A` to the i-th position of the chunk and denote the modified chunk as $\hat{c}$. In other words, $\hat{c}$ is the concatenation of $c_{\ldots i-1}$, `A`, and $c_{i\ldots}$.

The critical observation is that we can decompose the generation of the read r from $\hat{c}$ by a pHMM as follows (Fig. 2.4):

1. Generate $r_{\ldots j}$ and state s from $c_{\ldots i}$. The forward algorithm computes this probability ($F_s[i][j]$).

2. Generate the j-th base of the read or a gap symbol – at the newly introduced base `A`.

3. Generate $r_{j\ldots}$ or $r_{j+1\ldots}$ from $c_{i\ldots}$ and the state D or M. The backward algorithm computes this probability ($B_D[i][j]$ or $B_M[i][j+1]$).

[5] "removing errors in chunks" is a not well-defined problem because I do not **know** the ground truth for each chunk. In other words, there is no way to locate "errors" in a chunk.

Note that $F_s[i][j] = L(r_{\cdots j}, s|c_{\cdots i})$ where $r_{\cdots j}$ is the prefix of r up to the $j-1$-th base and $c_{\cdots i}$ is defined similarly. Also, $B_M[i][j+1] = L(r_{j+1\cdots}, |c_i \cdots, M)$ where $r_{j+1\cdots}$ is suffix of r from the $j+1$-th base and $c_{i\cdots}$ is defined similarly.

If the pHMM generates a base of the read, there are $|r|$ cases, i.e., $r_0, \cdots, r_{|r|-1}$. Otherwise, the pHMM generates a gap symbol, and there are $(|r|+1)$ cases in total, i.e., before the j-th base for $j = 0, \cdots, |r|-1$, and after the last base.

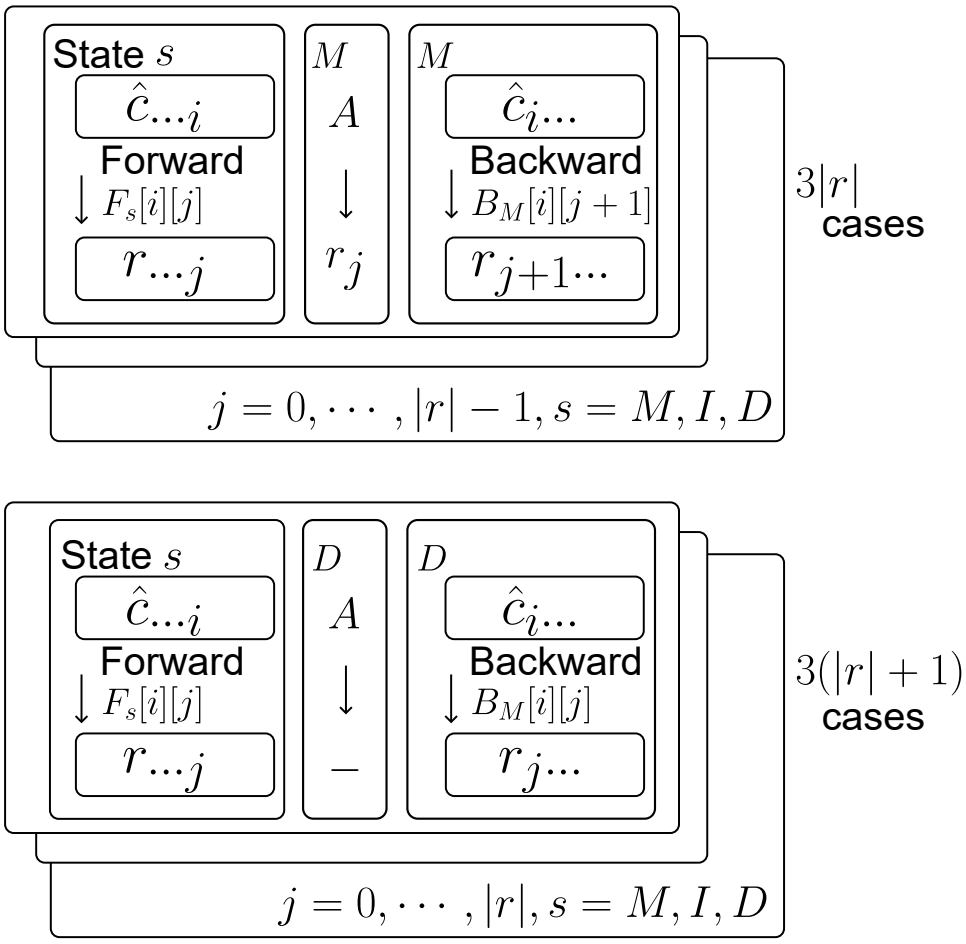

Figure 2.4: The decomposition of the process of generating a read r from an edited chunk $\hat{c}$. We can sum up the probabilities of these cases to get the probability of generating r from $\hat{c}$.

Suppose we have computed F and B by the forward and backward algorithm beforehand. Then, if the added base (A) matches a base at some position j in the read, the total likelihood is

$$\sum_{s,j} F_s[i][j] T_{s,M} P_M(r_j|\mathtt{A}) B_M[i][j+1] \tag{2.10}$$

Otherwise, i.e., if the introduced base matches a gap symbol, the total likelihood is

$$\sum_{s,j} F_s[i][j] T_{s,D} B_D[i][j] \tag{2.11}$$

As we have already computed the B and F, it takes $O(|r|)$ to compute the likelihood $L(r|\hat{c})$. I can generalize this argument to substitutions and deletions for all positions in the chunk.

In summary, let $\hat{c}_{i,e}$ be the chunk after introducing an edit operation e at the i-th position of the chunk. $L(r|\hat{c}_{i,e})$ is the sum of (2.10) and (2.11). Given N reads, it takes $O(N|c||r|)$ time to compute the following value $P[r][i][e]$ for each read r, each position i in the chunk, and each edit operation e:

$$P[r][i][e] = \ln L(r|\hat{c}_{i,e}) - \ln L(r|c) \tag{2.12}$$

I call P the *perturbation matrix*, as it tells how the likelihood changes by a perturbation, or an edit operation, on a chunk.

How can I use the perturbation matrix to take a consensus from reads or, equivalently, to maximize the total log-likelihood $\sum_r \ln L(r|c)$?

To search for a nearly maximum value of the total log-likelihood, I sum up P over reads, i.e., I compute $\sum_r P[r][i][e]$. If this value is positive, the total likelihood increases by changing c into $\hat{c}_{i,e}$. Thus, I edit the chunk c by (i, e) until there are no (i, e) such that $\sum_r P[r][i][e] > 0$ holds. I regard the resulting sequence as the consensus sequence from the reads.

Also, I use the perturbation matrix P to call variants between the copies of a chunk. For example, suppose there is an SNV at the i-th base on the chunk, and I can represent the SNV as (i, e). Then, ideally, half of the reads support this SNV, i.e., half of the reads satisfy $P[r][i][e] > 0$ in the perturbation matrix P. Conversely, if I have a sufficient number of reads with $P[r][i][e] > 0$, it suggests that (i, e) is a variant between the copies of the chunk.

Formally, I sum up P over r but ignore negative values and denote it as V:

$$V[i][e] = \sum_r \max(P[r][i][e], 0) \tag{2.13}$$

$V[i][e]$ represents how much the total likelihood we can improve by splitting reads into two clusters based on the variants at (i, e). I regard a set of (i, e) giving large $V[i][e]$ as variants.

However, this naive criterion suffers from false positive variants. I will describe how to eliminate these false variants in the next section.

2.3.5 Filtering out false positive variants

The variant calling explained above does not consider two crucial properties of error-prone reads.

First, I should consider the biases in the base callers on the strand direction. Specifically, long-read sequencers introduce more errors in the forward strand on specific DNA sequences than in the reverse strand, and vice versa. Thus, I sometimes observe a specific nucleotide that only occurs in the forward strand, which is often mistaken as a variant. I filter out the false positive variants due to this *strand bias* by inspecting the perturbation matrix P in the forward and reverse strands separately.

Second, variant calling should consider the number of reads supporting a variant. For example, suppose 12 out of 60 reads on a chunk support a variant. This variant can be an artifact that occurred in 12 reads by chance due to sequencing errors. To rule out this possibility, I simulate reads by a pHMM to calculate the probability that I mistakenly regard an error as a variant.

By considering these two sources of false positive variants, the called variants from the perturbation become very accurate.

In the following two sections, I will argue how to determine whether a variant on a chunk is a false positive. For simplicity, I fix a chunk c, the reads R aligned to it, the position i, and the edit operation e at i-th position on the chunk. In other words, I hereafter denote the changes of the log-likelihood of each read $r \in R$ as $p_r = P[r][e][i]$ to make equations simple. In addition, I refer to the variant on the chunk represented as (i, e) as *the variant*.

Checking bias in the strand directions

If the copy number of the chunk c is two, and there is only one variant, (i, e), I separate the reads into two clusters, $\{r \mid p_r > 0\}$ and $\{r \mid p_r \leq 0\}$. By separating the reads into two copies in this manner, the log-likelihood improves by:

$$\sum_{r \in R} \max(p_r, 0) \tag{2.14}$$

If there is a bias in the strand direction, the difference between the value of (2.14) in the forward or reverse directions would be large. Let $\overrightarrow{R}$ be the set of reads aligned in the forward direction. I calculate the improvement of the likelihood on the forward strand as $\sum_{r \in \overrightarrow{R}} \max(p_r, 0)$. Similarly, by defining $\overleftarrow{R}$ as the reads aligned in the reverse direction, the improvement on the reverse strand is $\sum_{r \in \overleftarrow{R}} \max(p_r, 0)$. Then, I define the difference between them as d.

$$d = \sum_{r \in \overrightarrow{R}} \max(p_r, 0) - \sum_{r \in \overleftarrow{R}} \max(p_r, 0) \tag{2.15}$$

If there is a strand bias, d would be far from zero. I check this condition by a permutation test as follows.

First, I simulate a null distribution of d by shuffling the aligned direction and computing (2.15) two thousand times. Namely, at each repetition, I randomly sample $|\overrightarrow{R}|$ reads from R, denote it as S, and compute $\sum_{r \in S} \max(p_r, 0) -$

$\sum_{r \notin S} \max(p_r, 0)$. If d is not in the 5%-95% interval of this null distribution, I regard the variant as a false positive. Two thousand times is usually sufficient to approximate the distribution.

Checking the number of supporting reads

There is another source of false positive variants besides the strand bias. Because of the high error rate, some reads often share the same error at the same position, which can be mistakenly regarded as a variant.

To discard these false positives, I estimate the probability p that a read supports a variant by chance. Specifically, I create three *mutated chunks*, each containing either an insertion, a deletion, or a substitution. Then, I generate N reads by Algorithm 3 from the chunk c and denote them as R'. For each mutated chunk $\hat{c}$, I count the fraction of generated reads with higher log-likelihoods on the mutated chunk than on the original chunk. Formally, I define the fraction p as follows:

$$p = \frac{|\{r \in R \mid L(r|\hat{c}) > L(r|c)\}|}{N} \tag{2.16}$$

($L(r|c)$ is the likelihood of generating a read r from a chunk c.)

I treat this probability p as the probability that a read supports an insertion, a deletion, or a substitution variant by chance. Thus, if I have M reads supporting the variant, I assume M follows the binomial distribution $\mathrm{Binom}(|R|, p)$. I filter out the variant if the p-value is above a significance level, $0.05/9|c|$, where 9 means the number of edit operations (four types of insertion, four types of substitution, and a deletion), and I use Bonferroni correction to keep the false positive rate low. I generate 1000 reads by default, as I find that generating 1000 reads is accurate and fast enough.

I remove most of the false variants by combining these two filtering functions with other naive filtering functions, such as removing variants in homopolymers. Empirically, as shown in Fig. 2.11b, I achieved almost optimal clustering by these filtering functions.

2.3.6 Read-vs-read alignments to polish clusterings

(See Section 2.2.4 for the clustering algorithm.)

The previous section explained how to find variants based on the perturbation matrices. However, long-reads are error-prone, and these errors can blur these variants in reads. As a result, these reads might not be assigned to the correct cluster. In this section, I will discuss how to resolve this issue.

Remember that a chunk can have adjacent chunks in the reads, and I carry out clusterings on these chunks independently. I can leverage the clusterings of chunks adjacent to the chunk to correct errors in the clustering on a chunk.

For example, suppose two chunks u and v are adjacent and share the same copy number K. Also, suppose these two chunks have the same set of reads R

aligned to them. After the clustering, I have clusters of R on u and v. I denote these clusters, or partitions of R, as $R_1(u), \cdots, R_K(u)$ on u and $R_1(v), \cdots, R_K(v)$ on v. Then, if I were confident that the clustering on the chunk u is 100% accurate and the clustering on v is erroneous, I could discard the clustering on v, and use u's clustering instead. In other words, I substitute $R_k(v)$ with $R_k(u)$ for each k.

However, in the actual cases, I am never confident that a clustering is 100% accurate, and u and v share only some of the reads R in general. Thus, the above argument does not work as-is fashion. In addition, I compare the clustering on only two chunks (u and v) in the above example. As reads are long and possibly have three or more chunks, it is desirable to consider clusterings as many as possible.

In summary, to remove errors in the clusterings, I need to handle three issues below:

- I need to know the accuracy of the clustering on a chunk.
- Adjacent chunks can have different sets of reads aligned to them.
- A read can be aligned to three or more chunks. It is desirable to consider all the clustering of these chunks.

To solve these issues, I align reads to each other not at the base level but at the chunk level. Fig. 2.5 gives the overview of my approach.

Intuitively, for the first issue, I devise a dedicated match score between two chunks for alignment so that I can measure the accuracy of clustering. Also, I allow alignments between reads to have gaps to consider the second issue. For the third issue, alignments can treat two or more chunks on a read.

These alignments between reads give *similarities* between reads, and I run a clustering algorithm on the similarity matrix to fine-tune clustering on the chunk (Fig. 2.5f).

Before explaining the detail of my method, I clarify my terminology. Specifically, I use the term "reads" in three meanings interchangeably so far:

- I have referred to the DNA sequences of reads as "reads."
- I have called the region of a read aligned to a chunk "a read aligned to the chunk".
- I have used the word "read" to denote the array of these occurrences of the chunks in a read.

Although I have made my argument simple by using the word "read" flexibly, from now on, this flexibility will be confusing because a chunk can be aligned to two different positions of the same "read" (e.g., the chunk v appears twice in Read1 in Fig. 2.5b).

Thus, I will use the terms defined as follows (Fig. 2.6).

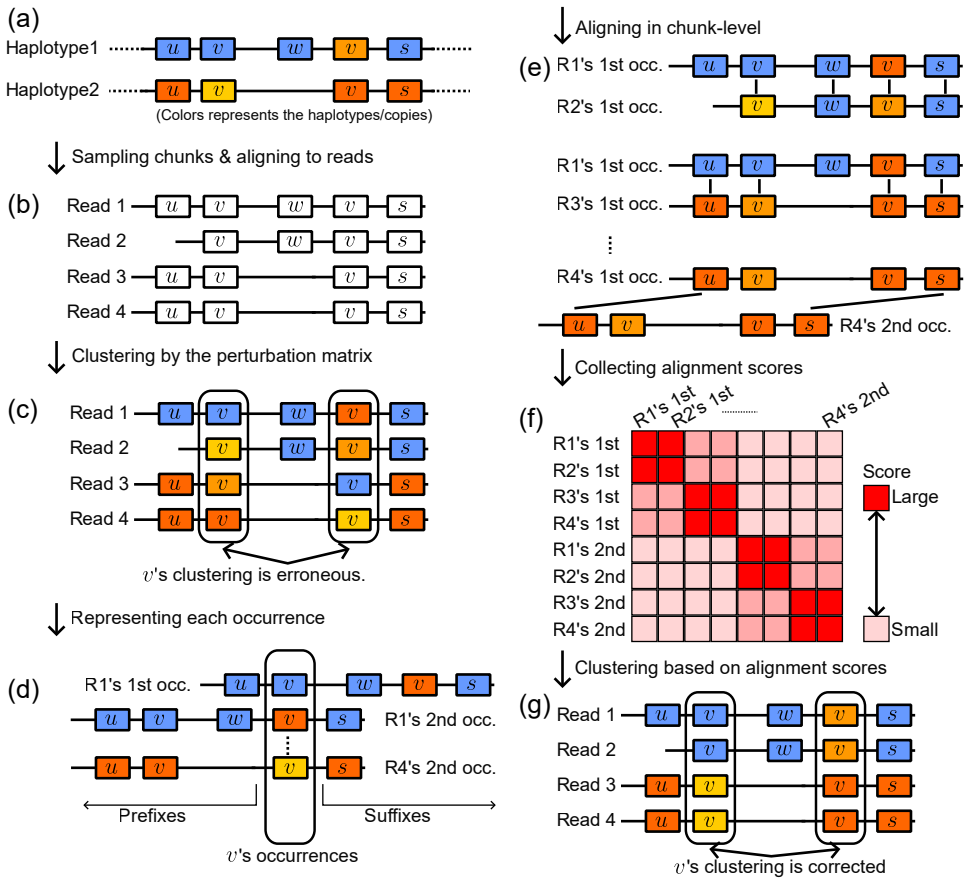

Figure 2.5: Schematic illustration of the read-vs-read alignments. (a): In this figure, I consider two haplotypes consisting of four chunks, u, v, w, and s. I color the chunks to distinguish the haplotypes and copies of these chunks. (b)-(c): Before clustering correction, I aligned these chunks to the reads and ran clusterings by the perturbation matrices on the chunks. Suppose that the clustering on v is erroneous due to the high sequencing error. (d): Before aligning the reads, I represent each of eight occurrences of v by a triple (the prefix, the occurrence, and the suffix). (e): I align these representations with each other by a dedicated match score (see Section 2.3.6). (f): The matrix of the alignment score. For example, the first occurrence of v in Read 1 and 2 aligns with a high alignment score. (g): The normalized spectral clustering on the matrix provides a new clustering of v, and I replace the old clustering with the new one.

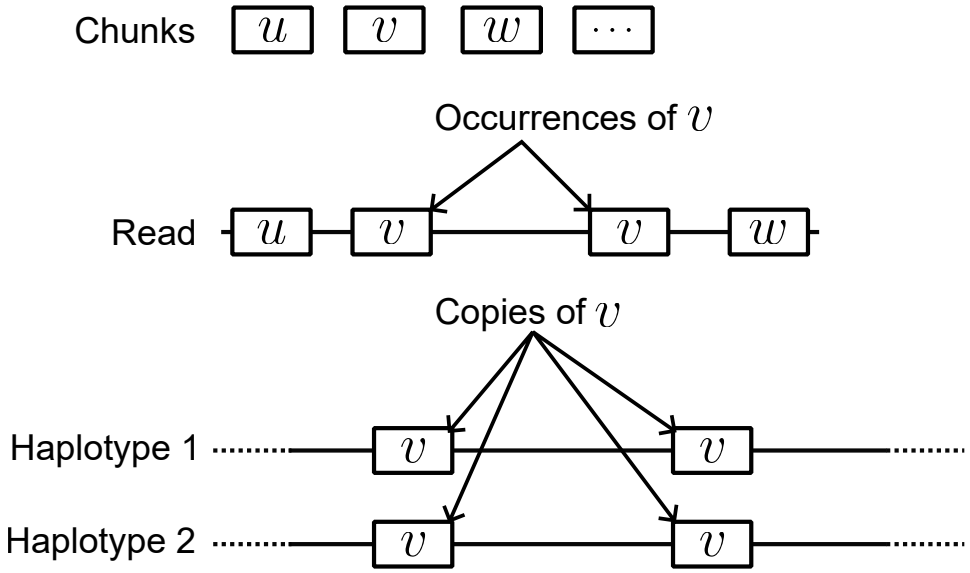

Figure 2.6: Definitions of occurrence and copies of a chunk.

- A *chunk* is a DNA sequence.
- The *copies* of a chunk are DNA sequences in the target region similar to the chunk. For example, I have four copies of v in the two haplotypes in Fig. 2.6.
- An *occurrence* of a chunk in a read is the sub-sequence of the read where I can align the chunk. For example, I have two occurrences of the chunk v in Read1 in Fig. 2.6.

I also use the term "read" to refer to the array of the occurrences of the chunks in a read. For example, Read in Fig. 2.6 is an array of occurrences of u, v, v, and w.

The definition of the alignment scheme is the most complicated part of this thesis. Thus, I decompose the explanation into three parts:

1. Defining the match score between two occurrences of the chunks.
2. Defining the gap score and alignments between reads.
3. Converting alignment scores into a similarity score to correct errors.

Defining match score

To define a scoring scheme between reads, I first need to define the *match score* between two occurrences in the reads. In other words, what is the match score between w in Read1 and s in Read2 in Fig. 2.5e? How about between w in Read1 and w in Read2 in Fig. 2.5e? As explained in this section, I compute the probability

that the two occurrences on the reads originate from the same genomic region. These two occurrences can "match" if this probability is greater than 0.5.

To compute this probability, remember that chunks represent DNA sequences, and I have removed overlapping chunks (Section 2.3.1). Thus, two different chunks must originate from different regions, and the alignment score between the two occurrences of them should be small. So, I define the match score between two occurrences of different chunks as $-\infty$. In other words, they never match in this alignment scheme regardless of the sequence similarity.

Next, I define the match score between two occurrences of the same chunk. To this end, let R be the occurrences of the chunk in the reads and the copy number of the chunk be K. I have run the clustering on this chunk by the perturbation matrix to partition these occurrences into K clusters, $R_1, \cdots, R_K$.

One obvious but not good approach for defining a match score is to use R_k directly. Namely, for r_1 and r_2 in R, I may define

$$\text{Match score}(r_1, r_2) = \begin{cases} 1 & \text{if } r_1 \in R_k \text{ and } r_2 \in R_k \text{ for some } k \\ 0 & \text{otherwise} \end{cases}$$

This match score fails to represent that different regions have different similarities. For example, suppose there are four chunks as in Fig. 2.7a. There are 1% differences between H1 and H2, while there are 0.1% differences between H2 and H3. Since the naive match score takes either 1 or 0, it fails to represent the similarities between these regions. In addition, this match score does not care about the errors in each occurrence.

To consider these two issues, I first relax the hard clustering, i.e., $R_1, \cdots, R_K$, to probabilistic representation. Namely, I will compute the probability that an occurrence $r \in R$ is in the k-th cluster based on the perturbation matrix P. This probability, denoted as $p_r[k]$, provides a way to define the match score.

For the reader's understanding, I give a concrete example of the occurrences, a perturbation matrix, and a partition of them (Fig. 2.7b).

To begin with, remember that a perturbation matrix represents how the log-likelihood of a read changes when I apply an edit operation to the chunk. I simplify these changes and use $1, -1$, and 0 to represent the case when the log-likelihood increases, decreases, or does not change, respectively.

Under this simplification, our example consists of seven occurrences $R = \{r_1, \cdots, r_7\}$ and the perturbation matrix defined as Eq. (2.17). Here, for simplicity, I assume the set of selected variants U has five pairs of (i, e), and I regard U as $\{1, 2, 3, 4, 5\}$. [6]

Further, suppose we have partitioned these occurrences into three clusters, $R_1 = \{r_1, r_2, r_3\}$, $R_2 = \{r_3, r_4\}$, and $R_3 = \{r_6, r_7\}$. I visualize this example in Fig. 2.7b.

[6] In the actual implementation, I have nine types of edit operations and convert $P(i, e)$ into $9i + e$, making $P[r][i][e]$ to be $|R| \times 9L$-dimensional matrix.

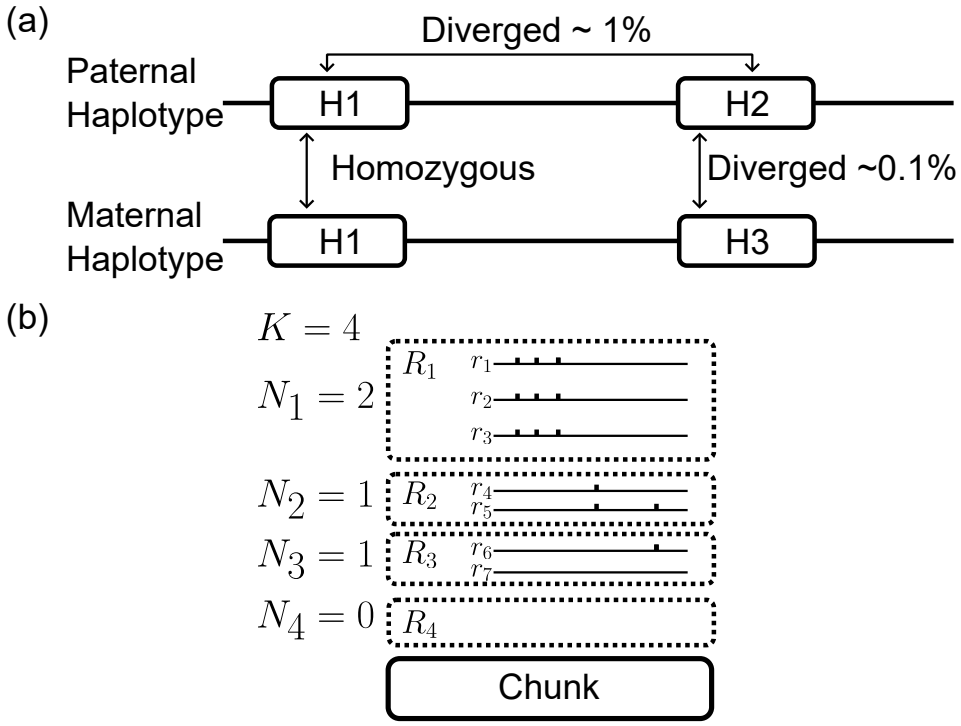

Figure 2.7: A schematic illustration of the match score. (a): If a chunk's copy number is four, four regions in the underlying genome correspond to these copies. Note that these regions can be homozygous. (b): The example of the perturbation matrix (Eq. (2.17)). The vertical ticks in the occurrences represent the variants. K is the copy number, and $N_k, k = 1, \cdots, 4$ is the estimated copy number of each cluster.

$$P = \begin{pmatrix} P[r_1] \\ P[r_2] \\ P[r_3] \\ P[r_4] \\ P[r_5] \\ P[r_6] \\ P[r_7] \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & -1 & -1 \\ -1 & -1 & -1 & 1 & 0 \\ -1 & -1 & -1 & 1 & 1 \\ -1 & -1 & -1 & -1 & 1 \\ -1 & -1 & -1 & -1 & 0 \end{pmatrix} \tag{2.17}$$

Intuitively, the first three columns of the perturbation matrix Eq. (2.17) represent the variant separating R_1 from other clusters, the fourth column characterizes R_2, and the fifth column separates R_2 and R_3 from R_1. Also, as errors in the fourth occurrence r_4 erased the fifth variant (i.e., $P[r_4][5] = 0$), the assignment of r_4 is less reliable than the assignment of the fifth occurrence, r_5. I integrate these intuitions into $p_r[k]$ by first expressing the difference between clusters $R_1, \cdots, R_K$ and then representing the sequencing errors.

To capture the difference between clusters, I sum up the perturbation matrix P over R_k. If the value, $\sum_{r \in R_k} P[r][i][e]$, is positive, the variant (i, e) increases the total log-likelihood in the k-th cluster.

In the running example,

$$\begin{matrix} R_1 \\ R_2 \\ R_3 \end{matrix} \begin{pmatrix} P[r_1] + P[r_2] + P[r_3] \\ P[r_4] + P[r_5] \\ P[r_6] + P[r_7] \end{pmatrix} = \begin{pmatrix} 3 & 3 & 3 & -3 & -3 \\ -2 & -2 & -2 & 2 & 1 \\ -2 & -2 & -2 & -2 & 1 \end{pmatrix} \tag{2.18}$$

The first, second, and third variants increase the log-likelihood in R_1.

I define these (i, e) as U_k and denote the chunk after introducing all the operations in U_k to c as $\hat{c}_{U_k}$. [7] Precisely,

$$U_k = \{(i, e) \in U \mid \sum_{r \in R_k} P[r][i][e] > 0\} \tag{2.19}$$

$$\hat{c}_{U_k} = \text{sequence after applying all elements in } U_k \text{ to } c. \tag{2.20}$$

In the running example, $U_1 = \{1, 2, 3\}$, $U_2 = \{4, 5\}$, and $U_3 = \{5\}$.

By writing $z_r = k$ to mean r is in the k-th cluster, I calculate the probability that the occurrence r is in the k-th cluster, $p_r[k]$, as follows:

[7] This might not be a valid sequence, as U_k can contain two operations at the same position – two substitutions at the same base of the chunk. However, I can avoid this issue by forcing each element (i, e) in U_k to be separated by at least D bases.

$$
\begin{aligned}
p_r[k] &= Pr\{z_r = k \mid r, c, \hat{c}_{U_1}, \cdots, \hat{c}_{U_K}\} && (2.21)\\
&= Pr\{r \mid z_r = k, c, \hat{c}_{U_1}, \cdots, \hat{c}_{U_K}\} \frac{Pr\{z_r = k \mid c, \hat{c}_{U_1}, \cdots, \hat{c}_{U_K}\}}{Pr\{r \mid c, \hat{c}_{U_1}, \cdots, \hat{c}_{U_K}\}} && (2.22)\\
&\propto Pr\{r \mid z_r = k, c, \hat{c}_{U_1}, \cdots, \hat{c}_{U_K}\} Pr\{z_r = k \mid c, \hat{c}_{U_1}, \cdots, \hat{c}_{U_K}\} && (2.23)\\
&\propto Pr\{r \mid z_r = k, c, \hat{c}_{U_1}, \cdots, \hat{c}_{U_K}\} && (2.24)\\
&= Pr\{r \mid \hat{c}_{U_k}\} && (2.25)
\end{aligned}
$$

Here, I use Bayes' theorem to obtain Eq. (2.22), drop a term independent from k to obtain Eq. (2.23), assume $Pr\{z_r = k \mid c, \hat{c}_{U_1}, \cdots, \hat{c}_{U_K}\}$ is a uniform distribution to obtain Eq. (2.24), and use $z_r = k$ to obtain Eq. (2.25).

Thus, I need $Pr\{r \mid \hat{c}_{U_k}\}$, the likelihood of r from $\hat{c}_{U_k}$, to compute the posterior probability, $p_r[k]$.

To this end, remember that $P[r][i][e]$ is how the log-likelihood changes when applying the variant (i, e) to the chunk c. Thus, I can approximate the change of the log-likelihood when applying all the operations in U_k to the chunk c by summing up $P[r][i][e]$ over (i, e) in U_k. I denote this value as $l(r|\hat{c}_{U_k})$. Formally,

$$
l(r|\hat{c}_{U_k}) = \sum_{(i,e) \in U_k} P[r][i][e] \tag{2.26}
$$

As a result, I approximate the log-likelihood of r in the k-th cluster. Precisely, given that the likelihood of an occurrence r from the chunk is defined as $L(r|c)$ (Section 2.3.3), $\ln Pr\{r \mid \hat{c}_{U_k}\} = \ln L(r|\hat{c}_{U_k}) \approx \ln L(r|c) + l(r|\hat{c}_{U_k})$.

In the running example, I have the following vector for r_1.

$$
\begin{pmatrix} l(r_1|\hat{c}_{U_1}) \\ l(r_1|\hat{c}_{U_2}) \\ l(r_1|\hat{c}_{U_3}) \end{pmatrix} = \begin{pmatrix} P[r_1][1] + P[r_1][2] + P[r_1][3] \\ P[r_1][4] + P[r_1][5] \\ P[r_1][5] \end{pmatrix} = \begin{pmatrix} 3 \\ -2 \\ -1 \end{pmatrix} \tag{2.27}
$$

Similarly, for r_4 and r_5, I have

$$
\begin{pmatrix} l(r_4|\hat{c}_{U_1}) \\ l(r_4|\hat{c}_{U_2}) \\ l(r_4|\hat{c}_{U_3}) \end{pmatrix} = \begin{pmatrix} -3 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} l(r_5|\hat{c}_{U_1}) \\ l(r_5|\hat{c}_{U_2}) \\ l(r_5|\hat{c}_{U_3}) \end{pmatrix} = \begin{pmatrix} -3 \\ 2 \\ 1 \end{pmatrix} \tag{2.28}
$$

Now, we are ready to compute $p_r[k]$, the probability that the occurrence r is in the k-th cluster.

$$
\begin{aligned}
p_r[k] &\propto Pr\{r \mid \hat{c}_{U_k}\} && (2.29)\\
&\approx \exp\left(\ln L(r|c) + l(r|\hat{c}_{U_k})\right) && (2.30)\\
&\propto \exp l(r|\hat{c}_{U_k}) && (2.31)
\end{aligned}
$$

Here, I remove $L(r|c)$ to obtain Eq. (2.31) because it does not depend on k.

I call $p_r[k]$ the *posterior probability* that the occurrence r is in the k-th cluster.

In the example (Eq. (2.17), Fig. 2.7), the posterior probability of the first occurrence r_1 is as follows:

$$\begin{pmatrix} p_{r_1}[1] \\ p_{r_1}[2] \\ p_{r_1}[3] \end{pmatrix} = \begin{pmatrix} e^3 \\ e^{-2} \\ e^{-1} \end{pmatrix} / \left(e^3 + e^{-2} + e^{-1}\right) \approx \begin{pmatrix} 0.97 \\ 0.01 \\ 0.02 \end{pmatrix} \tag{2.32}$$

Likewise, I can compute the posterior probabilities of r_4 and r_5.

$$\begin{pmatrix} p_{r_4}[1] \\ p_{r_4}[2] \\ p_{r_4}[3] \end{pmatrix} \approx \begin{pmatrix} 0.01 \\ 0.72 \\ 0.27 \end{pmatrix}, \begin{pmatrix} p_{r_5}[1] \\ p_{r_5}[2] \\ p_{r_5}[3] \end{pmatrix} \approx \begin{pmatrix} 0.00 \\ 0.73 \\ 0.27 \end{pmatrix}, \tag{2.33}$$

At the beginning of this section, I argued that r_4 should be less reliable than r_5. As we expected, while the probability that r_4 is in the first cluster R_1 is $p_{r_4}[1] = 1\%$, there is almost no chance that r_5 is in R_1 (i.e., $p_{r_5}[1] \approx 0$).

Now, I can compute the probability that two occurrences originate from the same clusters. Specifically, the posterior probability of r_2 is as follows:

$$\begin{pmatrix} p_{r_2}[1] \\ p_{r_2}[2] \\ p_{r_2}[3] \end{pmatrix} \approx \begin{pmatrix} 0.97 \\ 0.01 \\ 0.02 \end{pmatrix} \tag{2.34}$$

Thus, the probability r_1 and r_2 originate from the same cluster is the inner product of the posteriors:

$$\begin{pmatrix} p_{r_1}[1] \\ p_{r_1}[2] \\ p_{r_1}[3] \end{pmatrix} \cdot \begin{pmatrix} p_{r_2}[1] \\ p_{r_2}[2] \\ p_{r_2}[3] \end{pmatrix} = \sum_{k=1}^{3} p_{r_1}[k] p_{r_2}[k] = 0.97^2 + 0.01^2 + 0.02^2 \approx 0.94 \tag{2.35}$$

However, as depicted in Fig. 2.7a, there are two copies of H1, and two occurrences can come from these two different copies of H1. Thus, this probability differs from the probability that these two occurrences originate from **the same genomic region**. I need to consider the copy numbers of these clusters.

Specifically, the chance that two occurrences in the H1 share the same genomic region is $1/2$. Therefore, I can calibrate the probability by dividing $p_{r_1}[1]p_{r_2}[1]$ by two. The correct probability is $0.97^2/2 + 0.01^2/1 + 0.02^2/1 \approx 0.47$.

To generalize this idea, I denote $C_1, \cdots, C_K$ as the number of occurrences in the clusters, i.e., $C_k = |R_k|$. Also, let $\bar{c}$ be the average haploid coverage, which is determined as Eq. (2.2) when I determine the copy number of the chunks. Then, I estimate the copy numbers of clusters as $N_1, \cdots, N_K$ by minimizing the mean squared error defined below such that $\sum_k N_k = K$ (Fig. 2.7b).

$$\sum_k \|\bar{c}N_k - C_k\|^2 \tag{2.36}$$

Algorithm 6 Copy number (N_k) estimation for a clustering

Input: The size of the clusters $C_1, \cdots, C_K$ and the haploid coverage $\bar{c}$
Output: The copy number of clusters, $N_1, \cdots, N_K$

Initialize $N_1, \cdots, N_K$ as 0
for K times **:**
  Select k that reduces the mean squared error the most.
    $k \leftarrow \arg\max_k \|\bar{c}N_k - C_k\|^2 - \|\bar{c}(N_k + 1) - C_k\|^2$
    $N_k \leftarrow N_k + 1$
return $N_1, \cdots, N_K$

I obtain the optimal solution by a greedy algorithm (Algorithm 6).

Now, we have all materials needed to define the match score between two occurrences of the same chunk. Suppose we have the posterior probabilities of two occurrences of the same chunk, $p = (p_1, \cdots, p_K)$ and $q = (q_1, \cdots, q_K)$. These occurrences are on the same genomic region if they are in the same cluster (with a probability of $p_k q_k$ for each k), the cluster has at least one copy in the genome ($N_k > 0$), and they are in an identical copy (with a probability of $1/N_k$). I denote this probability as the *match probability* a:

$$a(p, q) = \sum_{k=1, N_k \neq 0}^{K} \frac{p_k q_k}{N_k} \tag{2.37}$$

I convert this match probability to *match score* by mapping the probability a to the log-odds-ratio, $\ln \frac{a}{1-a}$. [8]

To see the rationale behind this mapping, suppose $a(p, q) = 0.5$. Then, a is equal to $1 - a$. Thus, the probability that two occurrences originate from the same genomic region is the same as the probability that they do not. In this case, the match score is zero ($\ln \frac{0.5}{1-0.5} = 0$). Also, the log-odds-ratio is positive if and only if $a > 0.5$. Thus, it gives a reasonable match score because it prefers match probabilities more than 0.5.

In the running example, the match probability and score between r_4 and r_5 are:

$$a(p_{r_4}, p_{r_5}) = \frac{0.01 \cdot 0.00}{2} + \frac{0.72 \cdot 0.73}{1} + \frac{0.27 \cdot 0.27}{1} \approx 0.6$$
$$\ln \frac{a(p_{r_4}, p_{r_5})}{1 - a(p_{r_4}, p_{r_5})} \approx 0.4$$

In contrast, as the match probability between r_1 and r_5 is as small as 0.01, the match score is $\ln \frac{0.01}{0.99} \approx -4.3 < 0$.

[8] Note that if $a = 0$ or $a = 1$, the log-odds-ratio is undefined. To avoid $a = 0$, I convert a to $\max(a, 10^{-20})$. Similarly, to avoid $a = 1$, I convert a to $\min(a, 1 - 10^{-20})$

As expected, the match score prefers the match among the same cluster, i.e., r_4 and r_5, than the match between different clusters, i.e., r_1 and r_5.

In summary, the match score between two occurrences (r_1 and r_2) is defined as follows:

$$\text{Match score}(r_1, r_2) = \begin{cases} -\infty & \text{if they represent two different chunks} \\ \ln \frac{a}{1-a} & \text{otherwise} \end{cases} \tag{2.38}$$

In the next section, I represent gaps in the reads and derive the complete alignment scheme to correct errors in the clusterings.

Define gaps scores and alignments between two reads

Because of the high error rate in the reads, I sometimes can not align a chunk to a read, making a "gap" in the reads (Fig. 2.8a). Thus, to consider gaps caused by highly erroneous regions, I need to allow gaps in chunk-level alignments.

In contrast, suppose the genome has a large heterozygous insertion (Fig. 2.8b). Chunks derived from this insertion appear only in one haplotype and create large insertions in the reads at the chunk level.

These two types of gaps are different because I want to allow only the first case of these two. Thus, I introduce the gap open penalty G and the gap extension penalty E. By setting G nearly zero and E much smaller than G, I allow one-length gaps caused by sequencing errors and inhibit large gaps caused by structural variants. After testing several parameters, I set $G = -0.5$ and $E = -100$ as default values.

The last thing to consider is the multiple occurrences of the same chunk in the same read. Specifically, suppose a read has two occurrences of the same chunk (e.g., Read1 in Fig. 2.5 has the chunk v twice). To distinguish these two occurrences, I first locate the position of the occurrence in the reads. Then, I split the reads into three components – the prefix before it, the occurrence itself, and the suffix after it. For example, I convert two occurrences of v in Read 1 in Fig. 2.5 as follows:

1. The first occurrence is converted to the prefix (u), the occurrence itself, and the suffix (w, v, s).
2. The second occurrence is converted to the prefix (u, v, w), the occurrence itself, and the suffix (s).

Given two of these representations, I align the prefixes to each other, the occurrence to each other, and the suffixes to each other by the alignment scheme (Eq. (2.38)), G, and E).

For given N occurrences of a chunk, these alignments provide an $N \times N$-dimension matrix A, where the (i, j)-element is the alignment score between the i-th and the j-th occurrences. In the next section, I will discuss how to convert this matrix A into a similarity matrix to carry out clustering.

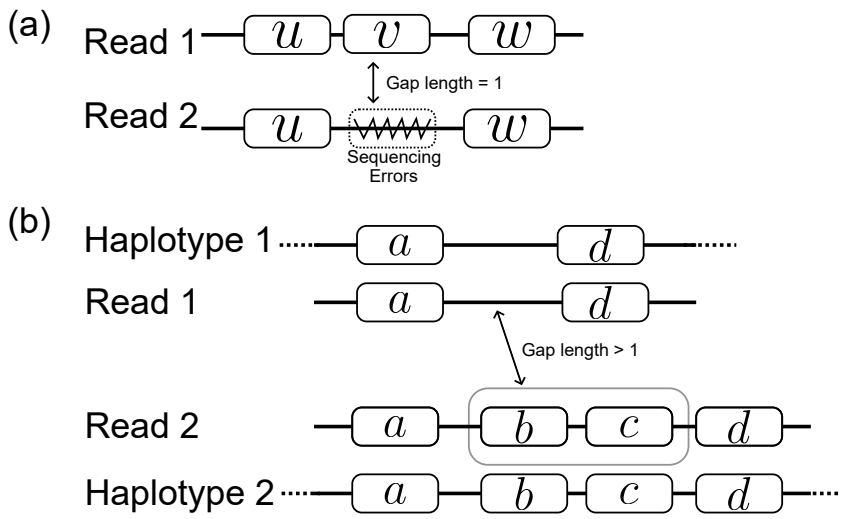

Figure 2.8: Schematic illustration of the two types of gaps in the alignments at chunk-level. (a): The gap created by high sequencing errors in the reads. The chunk v is absent from the second read. (b): The gap created by a large structural variant between the haplotypes. The two chunks (b and c) are absent from the first read. I only allow alignments from the first case by setting the gap extension penalty much smaller than the open penalty.

Convert an alignment score into a similarity score

The final step is to find the clusters from the alignment scores between reads.

To this end, remember that I define the match score as the log-odds-ratio of the alignment probability. Further, I assume that the alignment score between two reads is the log-odds-ratio of the *overlapping probability*. Formally, I interpret the matrix A as

$$A[i][j] = \text{alignment score between } r_i \text{ and } r_j \tag{2.39}$$

$$= \ln \frac{P\{r_i \text{ aligns } r_j\}}{1 - P\{r_i \text{ aligns } r_j\}} \tag{2.40}$$

I solve this equation to get a $N \times N$-dimension similarity matrix S.

$$S[i][j] = P\{r_i \text{ aligns } r_j\} = \frac{1}{1 + \exp(-A[i][j])} \tag{2.41}$$

As illustrated in Fig. 2.9, this conversion maps large alignment scores to near one and small scores to near zero.

Finally, I run the spectral clustering on S (Algorithm 7) and replace the original clustering $R_1, \cdots, R_K$ with them. I select the normalized spectral clustering because it handles the similarity matrix well. See ([145]) for further rationale.

In summary, I correct errors in the clusterings in four steps.

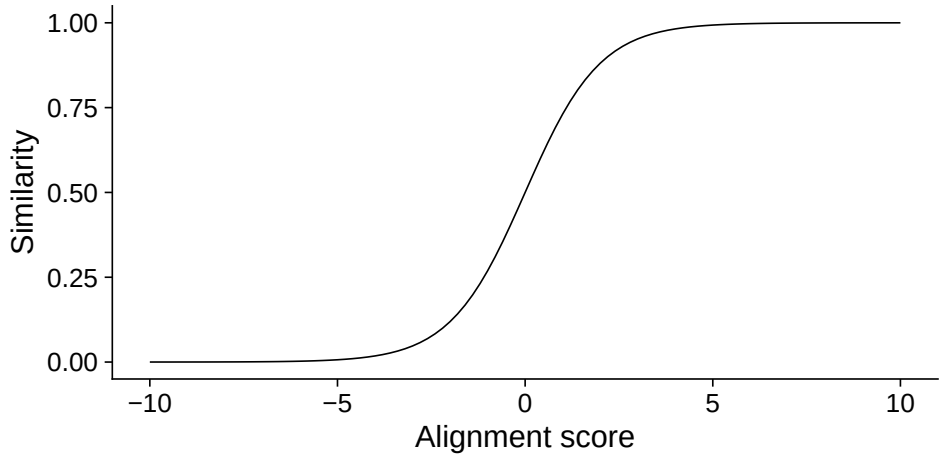

Figure 2.9: The function to convert the alignment score into similarity.

Algorithm 7 The spectral clustering on a similarity matrix.

Input: $N \times N$ similarity matrix S and an eigenvalue threshold t(=0.25 by default)
Output: Clustering of N elements

1: Let D be a diagonal matrix with $D_{i,i} = \sum_j S_{i,j}$ ▷ Row sum
2: $L \leftarrow I - D^{1/2}SD^{1/2}$
3: $(v_1, \cdots, v_k) \leftarrow$ eigenvectors with eigenvalue smaller than t. ▷ There are k clusters
4: $V \leftarrow$ Combine $v_1, \cdots, v_k$ to make an $N \times k$ matrix.
5: Regard rows of V as k-dimensional feature vectors for elements.
6: Clustering these feature vectors by k means clustering.

1. For each chunk, I convert occurrences R of the chunk into posterior probabilities by Eq. (2.31).

2. For each chunk v,

   (a) I represent each occurrence of the chunk v in the reads into three components – the prefix up to it, the occurrence itself, and the suffix from it.

   (b) I align them with each other by the scoring scheme defined by Eq. (2.38), $G(= -0.5)$, and $E(= -100)$ to get alignment scores A.

   (c) I convert the matrix of alignment score A into the similarity matrix S by Eq. (2.41).

   (d) I carry out the spectral clustering on S to obtain new clustering $R_1, \cdots, R_K$, and replace the original clustering of v with them.

The posterior probabilities consider the similarities between clusters and errors in the reads. Also, the chunk-level alignments consider the long-range information given by the reads. Thus, this approach utilizes the sequence and structural information that the reads give us.

2.3.7 Serializing the separated copies

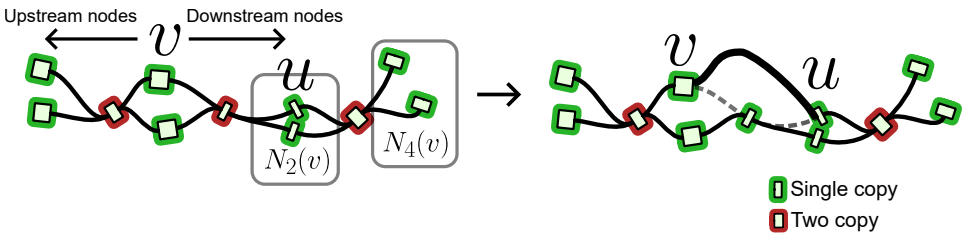

Figure 2.10: A schematic illustration of the graph simplification. The two graph boxes show the $N_2(v)$ (right). If u and v are foci for each other, I remove a path and add an edge between u and v (left).

The clustering aims to phase the chunks to obtain a fully phased assembly. However, there are long homozygous regions, i.e., regions without any variants. The clustering based on variants can not separate the copies of a chunk in these regions. Thus, I need an additional algorithm to solve these completely homozygous regions to produce a fully phased assembly.

Here, very long reads are helpful. Intuitively, if a read spans a homozygous region and connects phased regions, I can merge them. To handle a few remaining errors in clusterings, I develop merging criteria based on a statistical method.

I first construct a graph called *phased chunk graph* with the clustering information on the chunks in an analogous way to the *chunk graph* in Section 2.3.2.

Precisely, suppose that we have partitioned each chunk c into its K_c copies. Then, I have K nodes $(c, 1), \cdots, (c, K_c)$ in the graph for each chunk c. I call two nodes (c, k) and (c', k') *adjacent* if they satisfy the following four conditions:

- There is a read r such that c and c' are aligned to it.
- There are no other chunks between c and c' in r.
- The occurrence of c in r is in the k-th cluster in the c's clustering.
- The occurrence of c' in r is in the k'-th cluster in the c''s clustering.

Like the chunk graph in Section 2.3.2, I introduce H and T to represent the first (head) and last (tail) bases of a copy of a chunk.

By using these components, I define a *phased chunk graph* $G = (V, E)$ as follows:

- The set of nodes V is the union of $\{c_1, \cdots, c_K\}$ for each chunk c.
- The edge $E \subseteq (V \times \{H, T\}) \times (V \times \{H, T\})$ contains an edge between (v, p) and (u, q) if and only if v and u are adjacent and the p position of v and the q of u are connected by a read.
- Rigorously, I should define $V \times \{H, T\}$ as the nodes, but for simplicity, I refer to the set of copies of the chunks as V.

On a phased chunk graph, I estimate the copy numbers of each node $v = (c, k)$ using the same algorithm explained in Section 2.3.2. I define the upstream nodes of a node v in the graph as the nodes that are reachable from the head position of v (Algorithm 8). Also, the downstream nodes of v are the nodes reachable from the tail position of v.

Then, let v be a node adjacent to a homozygous node, and suppose we are finding the node downstream of v that we can merge with v.

To this end, let $N_d(v)$ be the set of nodes at a distance d downstream of v (Fig. 2.10), $N = |N_d(v)|$, and R be the number of reads that contain v and at least one of the nodes in $N_d(v)$. This method aims to check whether we can merge v and a node in $N_d(v)$.

Suppose there are R_u reads containing both v and $u \in N_d(v)$. If there were no errors in the clustering, I could span from v to u whenever R_u is greater than zero, i.e., whenever there are spanning reads. However, in reality, there are a small number of errors. Thus, I check whether R_u is statistically significant by using two binomial distributions $\mathrm{Binom}(R_u | R, p)$ with different values of p.

The first distribution represents the null hypothesis where a read from v reaches a node in $N_d(v)$ with an equal probability, i.e., R_u follows the binomial distribution with parameter $p = \frac{1}{N}$. Under this model, the likelihood of seeing R_u reads at $u \in N_d(v)$ is as follows:

Algorithm 8 Enumerating reachable nodes from a given node

Input: Phased chunk graph G, node v on G, start position p (= Head or Tail), and the max distance $D \in \mathbb{N}$

Output: The reachable nodes $N(v) = N_1(v), \cdots, N_D(v)$ from the position p of v with distance $d \leq D$.

$N_1(v), \cdots, N_D(v) \leftarrow ([\ ], [\ ], \cdots, [\ ])$ ▷ Initialize by the empty arrays.
$N_0(v) \leftarrow [(v, p)]$
for $d = 1, \cdots, D$ **:**
    for $(x, s) \in N_{d-1}(v)$ **:**
        for each edge $e = (x, s) \leftrightarrow (y, t)$ in G **:**
            if t is Head **:** ▷ Go to (y, t) then to the opposite side of y
                Push (y, Tail) to $N_d(v)$.
            else
                Push (y, Head) to $N_d(v)$.
Remove the second element, $(\_, p)$, from $N_1(v) \cdots, N_D(v)$.
return $N_1(v), \cdots, N_D(v)$

$$P_{\text{null}}(R_u|R, N) = \binom{R}{R_u}(\frac{1}{N})^{R_u}(1 - \frac{1}{N})^{R-R_u} \tag{2.42}$$

The second distribution represents the alternative hypothesis where almost all the reads passing v also pass u. Precisely, let ϵ be a small value, 0.1 by default, and define the likelihood as follows:

$$P_{\text{alt}}(R_u|R, N) = \binom{R}{R_u}(1 - \epsilon)^{R_u}\epsilon^{R-R_u} \tag{2.43}$$

I compare these two models by log-likelihood and denote it as $L_d(u|v) = \ln P_{\text{alt}} - \ln P_{\text{null}}$. I define the node and the distance giving the maximum $L_d(u|v)$ as the *focus* of v:

$$\text{focus}(v) = \arg\max_{d,u} L_d(u|v) \tag{2.44}$$

To span the homozygous region from v with minimum errors, I search a distance d and a node u such that

$$\text{focus}(v) = (d, u) \tag{2.45}$$
$$\text{focus}(u) = (d, v) \tag{2.46}$$

If there is any such (d, u), I eliminate an arbitrary path and draw an edge between v and u. I also decrement the copy number of the node in the eliminated path by one.

This simple statistical method accurately separates long homozygous regions by utilizing long reads and enables us to produce fully phased assemblies.

2.4 Results

2.4.1 Accuracy of phasing algorithm

I benchmarked the phasing algorithm (Algorithm 2) on simulated datasets. Specifically, I generated sequences of L bp in size (L =1K, 2K, 4K) and randomly introduced one edit operation to make the haplotypes to be phased. Then, I simulated noisy reads with varying error rates and coverages from these synthetic haplotypes. Finally, I haplotyped these reads and measured the accuracy by the Rand index ([121]). I also haplotyped by the exact algorithm instead of MCMC to measure the efficiency of the proposed heuristics.

The Rand index was around one minus the error rate across datasets(Fig. 2.11b, left). Algorithm 2 also showed the Rand index of more than 0.8 when phases were more than two (Fig. 2.11b, right). In addition, the correlation of the achieved value of Eq. (2.1) between the exact algorithm and the MCMC algorithm was more than 0.99. These results showed the robustness of the proposed method to errors and coverages and suggested that haplotyping of noisy reads can be done efficiently with only one variation between homologous chromosomes.

2.4.2 Comparison of assemblies on simulated reads

I assessed the performance of `JTK` by using simulated long reads. To evaluate the pipeline and software, I generated ground truth haplotype sequences using `NanoSim` ([159]) version 3.0 trained with in-house whole genome sequencing. I used `minimap2 -x asm5 --secondary=no --eqx -c` ([74]) to align assembled contigs to the ground truth haplotypes.

First, I benchmarked on simulated haplotypes. Precisely, to simulate large SVs and sporadic SNVs between haplotypes, I synthesized a 1Mbp random sequence and introduced one 50Kbp insertion, one 50Kbp deletion, and 0.1% divergence to create a synthetic haplotype.

`JTK` assembled 95% of the original genome in two contigs without errors from 60x ONT reads. The remaining five percent corresponds to both ends of the contigs with low coverage, like the two short contigs in the rightmost graph in Fig. 2.11a.

Similarly, `PECAT` output completely phased two contigs with a few errors near both ends of the contigs. These errors were presumably due to the low coverage of the reads.

Nonetheless, the phasing approach, i.e., `LongPhase` followed by `Flye`, produced low-quality phased assemblies. Precisely, although `LongPhase` could phase the long reads into two haplotypes by using one of the haplotypes as the reference, I observed large deletions and insertions in the contigs (Fig. 2.12).

Also, the traditional haploid assemblers (`Canu` and `Flye`) produced suboptimal assemblies. Specifically, `Flye` produced squished assembly representing one of

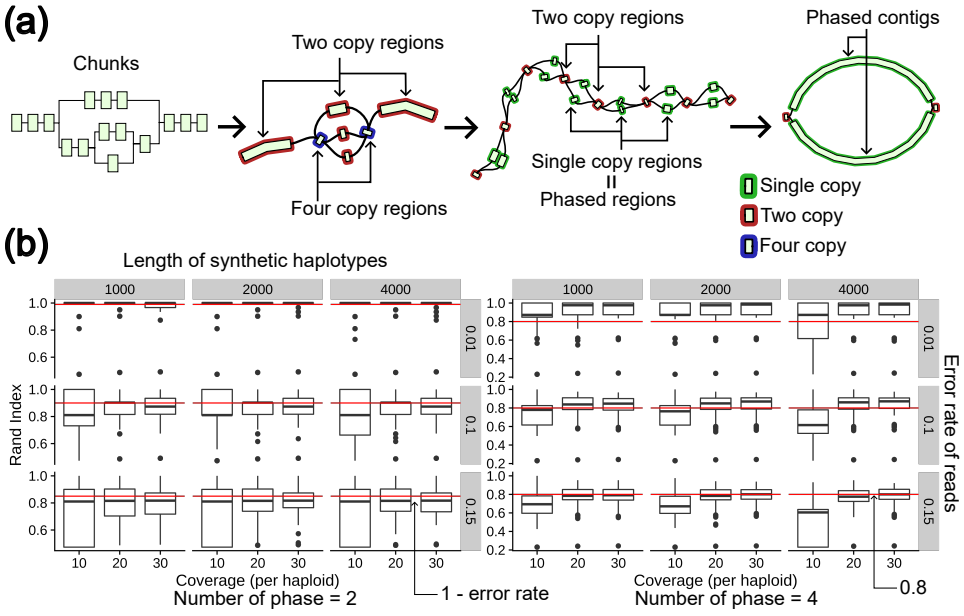

Figure 2.11: (a): Schematic illustration of how JTK iteratively improves the intermediate assembly graph. JTK checks the connection between chunks and then constructs an intermediate assembly graph by compressing simple paths (the first graph to the second). Then, JTK separates the chunks into their copies to generate a partially-phased assembly (the second to the third). Finally, it unzips the two copy regions to get the final assembly. I used `Gfaviz`([42]) for visualization. (b): The result of the clustering algorithm. The number of copies was two (the left) and four (the right). The red horizontal lines in the figures indicate 1− error rates in the left figure and 0.8 in the right.

the haplotypes while Canu assembled partially phased contigs.

Lastly, Phasebook generated as many as 55 contigs with the size 12 Mbp in total. Given that I created two 1-Mbp haplotypes, it suggests that Phasebook produced redundant and fragmented contigs. One explanation is that Phasebook was too sensitive to sequencing errors and mistakenly considered these errors as SNVs between haplotypes.

Next, I benchmarked on simulated haplotypes with large segmental duplications. Specifically, I synthesized a 1Mbp random sequence S, and joined two S by 200Kbp random sequence x, i.e., SxS. I then introduced 0.1% divergence to this haplotype to create synthetic diploid sequences with a large segmental duplication. From this synthetic genome, I simulated 60-fold ONT reads.

On this dataset, JTK assembled 95% of the original genome in two contigs without any errors (Fig. 2.13).

In contrast, other software programs, Flye, PECAT, and LongPhase+Flye, squished the difference between haplotypes. Moreover, Flye could not segregate the large segmental duplications. As a result, the Flye's contigs were as short as 500 Kbp, four times shorter than the ground truth haplotypes. Also, similar to the previous example, Phasebook produced many fragmented assemblies (186 contigs).

To further validate the proposed approach, I extract the COX and PGF haplotype of the MHC region from the human genome build 38 (hg38) as in the previous study ([84]). After removing all 'N' characters in both contigs, I simulated 60x long-reads by NanoSIM.

Out of four tested software, JTK and PECAT output two haplotype-phased contigs. I assessed the quality of these assemblies by QV, which is defined as $-10 \times \log_{10}(\text{Error})$ where Error is the number of errors between the assembled contigs and the ground-truth divided by the length of the assembly. These QVs were 62.0 and 59.1, respectively.

In contrast, LongPhase phased the reads into one haplotype block throughout the entire region, but Flye on each set of phased reads produced eight sub-optimal contigs. Also, there are switch errors, and the base-level QV of the assembled contig was around 20.

Phasebook produced 423 highly fragmented contigs, and the N50 was about 230Kbp. The quality values were around 20 across contigs, as Phasebook uses Flye as an assembly module.

Note that diploid assembly in this region is impossible only by HiFi reads because HiFi reads are usually shorter than 30Kbp. Seven homozygous regions are longer than 15Kbp without any variants between COX and PGF haplotypes (the longest one is 64Kbp).

Although JTK's performance comes at the expense of speed, JTK accurately assembles diploid sequences of up to several Mbp only with noisy ONT reads.

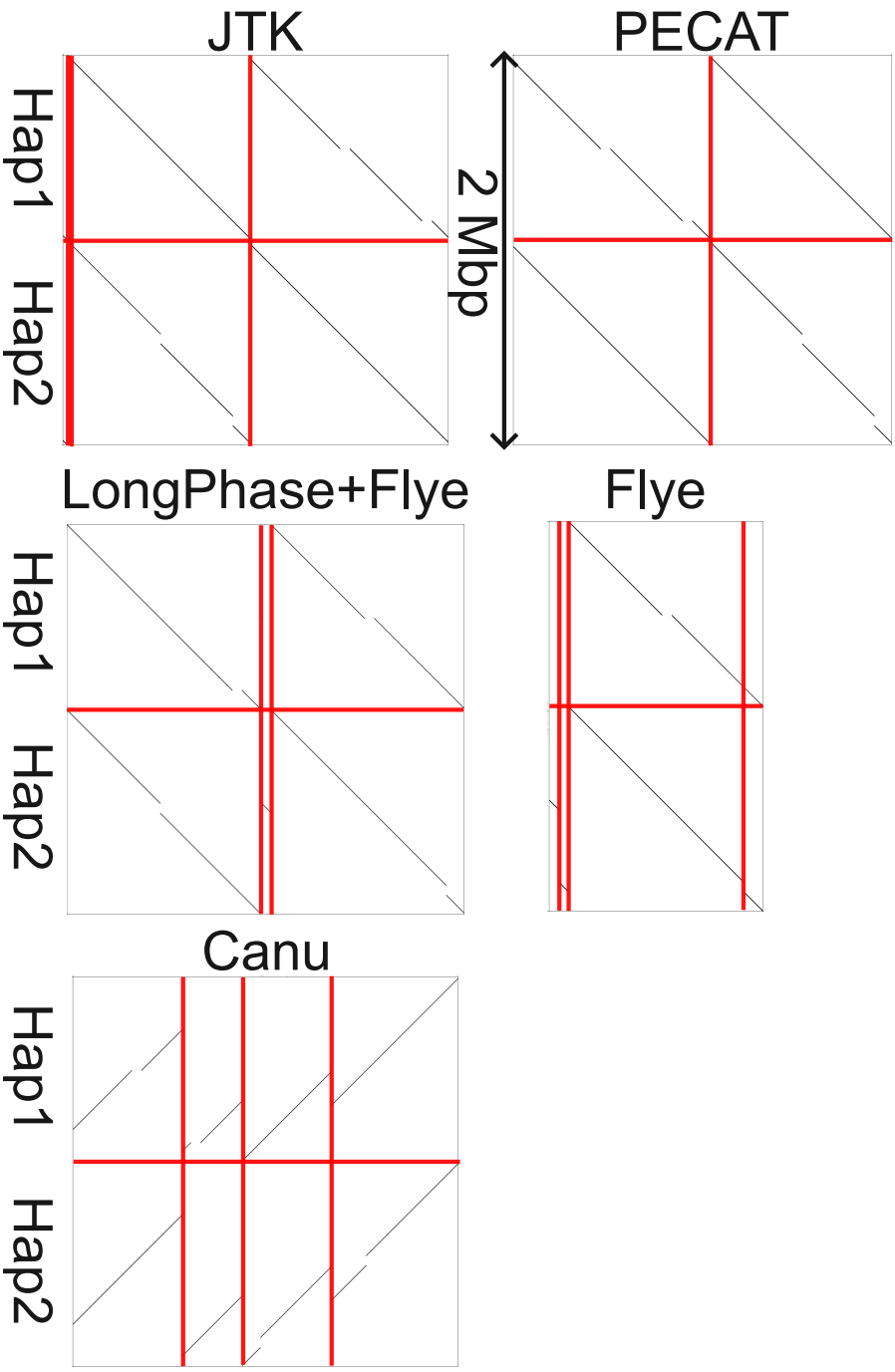

Figure 2.12: Doplots between the assemblies (x-axis) and the ground-truth haplotypes(y-axis). The ground-truth haplotypes contain two large (~ 50Kbp) SVs and 0.1% sequence divergence.

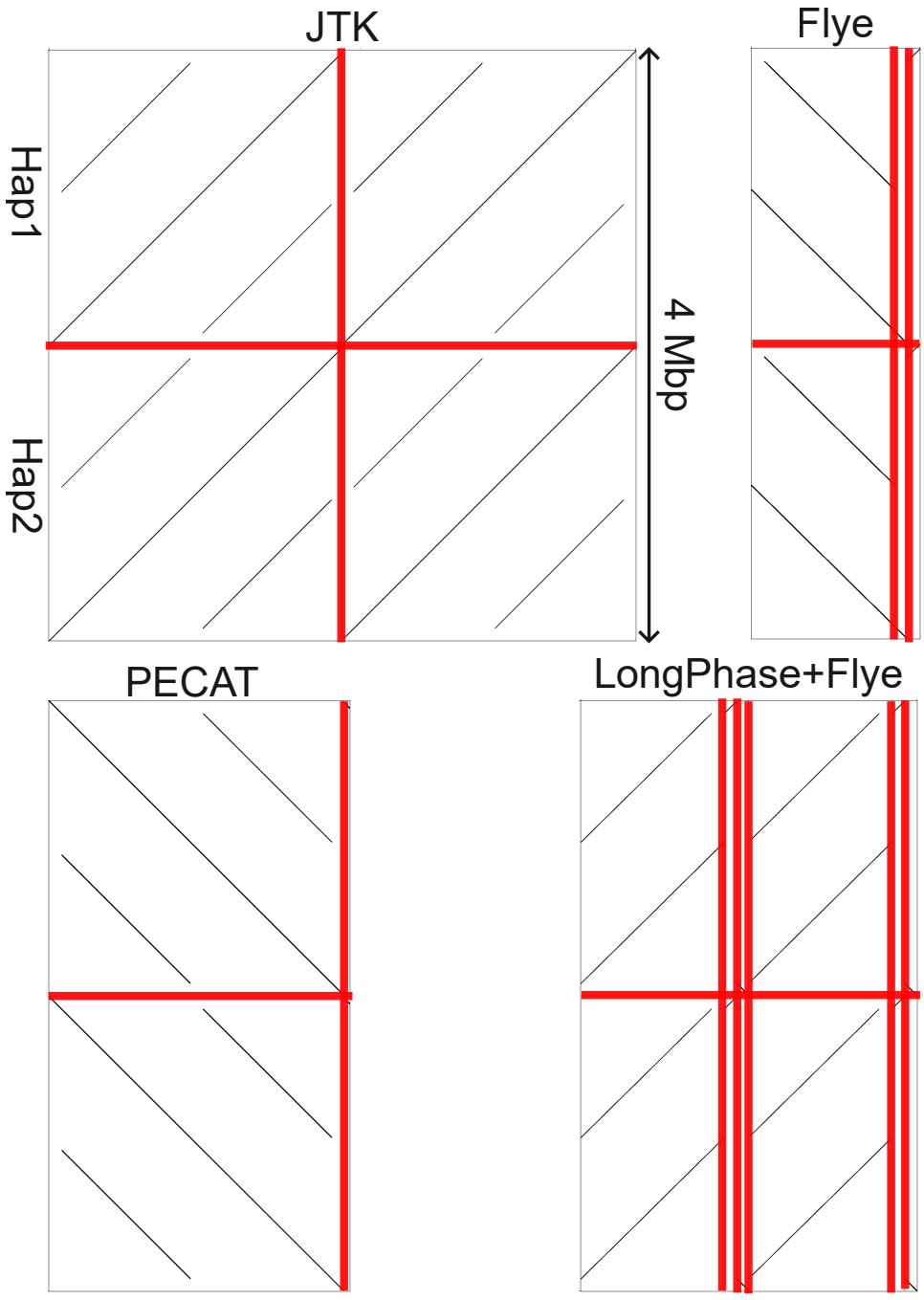

Figure 2.13: Dotplots between the assemblies (x-axis) and the ground-truth haplotypes (y-axis) with large segmental duplications.

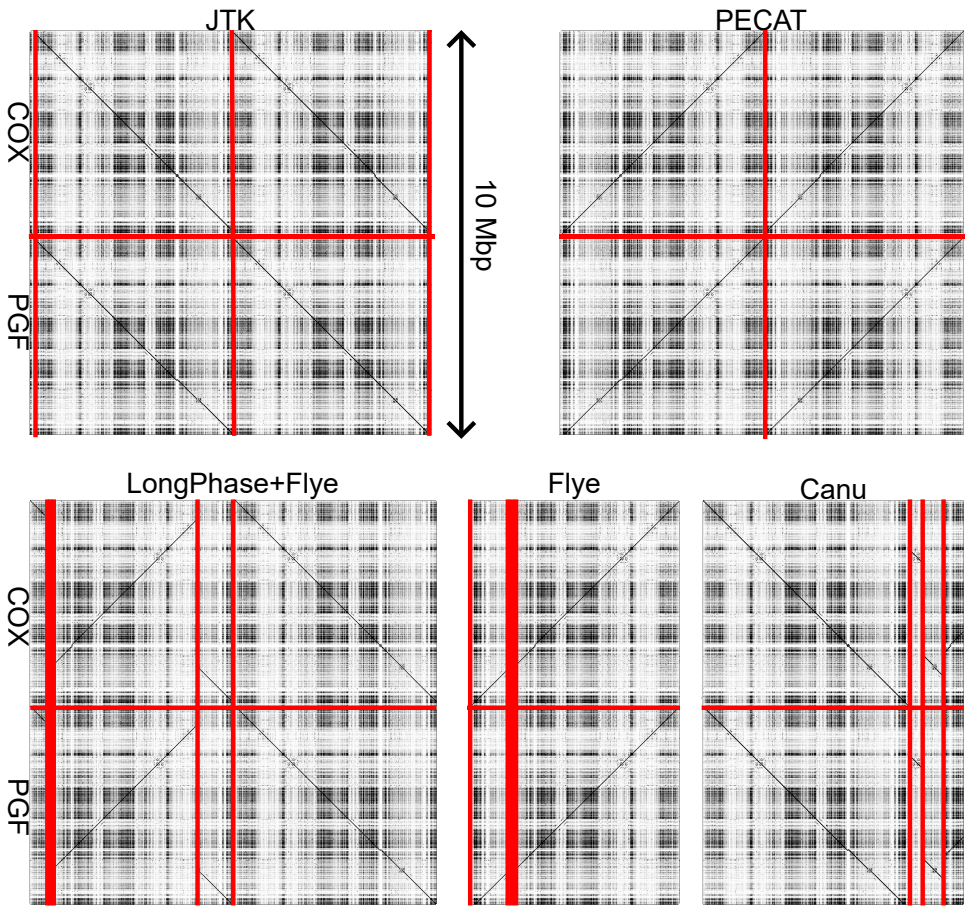

Figure 2.14: Dotplots between the assemblies (x-axis) and COX-PGF haplotypes (y-axis)

Software	# of contigs	Total bases (Kbp)	QV	Time (min)
JTK	2	9,502	62.0	40.6
PECAT	2	9,736	59.1	9.6
LongPhase+Flye	8	9,816	20.6	29.0
PhaseBook	423	75,357,275	-	31.5

Table 2.1: Results of the assembly on the reads simulated from the composition of the COX and PGF haplotypes.

2.5 Discussion and conclusion

2.5.1 Discussion

Genome assembly, especially diploid genome assembly, has been considered challenging when we can only use noisy long reads.

In this chapter, I introduced the regional diploid genome assembler, JTK, which aimed to fulfill this task with 60-fold long reads. JTK does not use alignments between reads and a string graph. Nor does it use k-mer and de Bruijn graph. Instead of these two traditional approaches, JTK uses kilobase-scale sequences called chunks. By detecting variants on these chunks and surveying the presence or absence of these chunks in the haplotypes, JTK assembles the genome in the diploid resolution.

I validated and compared my software with other programs on the three synthetic datasets with the ground-truth assemblies. JTK could accurately treat SVs, SDs, and SNVs at the same time and produced completely phased contigs on these datasets, which could not be achieved by other assemblers or software.

Currently, JTK can handle only megabase-scale regions because the pair-hidden Markov model spends a long time to call the variants on the chunks. Nonetheless, given that the hard-to-assemble regions in a genome comprise small fractions of the entire genome, JTK is a promising program to elucidate the diversity of these regions deeply.

2.5.2 Conclusion

In this chapter, I presented JTK, a regional diploid genome assembler and tested it based on the three artificial datasets of long reads. By leveraging kilobase-scale chunks, JTK could assemble complex genomes that other programs can not correctly reconstruct. However, these results were based on the synthetic reads generated by the NanoSIM software, and it is still unclear whether JTK can actually assemble genomes from the real data sets, i.e., ONT's or PacBio's reads. In the following section, to validate the performance of JTK, I examine the performance of JTK on mitochondrial genomes in plants and the MHC and LRC regions in the human genome.

Chapter 3

The landscape of the mitochondrial genome in eight strains of *Arabidopsis thaliana*

Abstract

Mitochondrial genomes in plants recombine frequently; even two strains in the same plant species have different structures of mitochondrial genome. Also, in some species, such as tomatoes and the model organism (*A.thaliana*), the mitogenomes encode genes responsible for the inability to create mature pollen, which makes the plants' mitogenomes important in agriculture.

As the sizes of the plant mitogenomes range from several hundred to a few mega-base pairs, long-reads technologies have produced assemblies of the reference mitogenomes for crop species. Nonetheless, as there can be high structural diversity among strains in one species, the reference may fail to represent the structures of the mitogenomes of a sample. Thus, we need an alternative way to investigate the structural diversity among strains.

Here, I used `JTK` to produce assemblies for nine datasets on the model organism, *A.thaliana* sequenced by PacBio sequel sequencers. `JTK` produced a fully resolved assembly for each strain. The base-level accuracies of these assemblies were approximately more than 99.99%, enabling me to compare the rate of recombinations and mutations. Also, `JTK` revealed "minor" structures in the samples through the alignments of the long-reads to the assemblies.

I confirmed that the locations of the recombination between the strains were near repeats. Comparison to the nucleus DNA suggested that the substitution rate in the mitogenome is at most 18 times slower than that in the nucleus, while the recombination rate is at least three times more frequent. These results hinted that the mitochondrial genomes in plants have a different way of repairing DNA damages or errors in DNA replication.

By using JTK, I could represent the mitogenome of a sample by a tailored assembly and other minor conformations. As the cost of sequencing decreases, assemblies on a larger scale would be possible, revealing a more detailed landscape of the mitogenome of a plant.

Code availability

The codes used in the analysis are available at https://github.com/ban-m/mito_check. The datasets used and the results in this section, including the assemblies, are available at https://mlab.cb.k.u-tokyo.ac.jp/~ban-m/dthesis/mitogenome.tar.gz. These files are also available upon request (banmasutani@gmail.com).

3.1 Introduction

Since Lynn Margulis confirmed with substantial evidence that mitochondria originated from external bacteria ([125]), mitochondria genomes, called mitogenomes, have been extensively investigated.

The mitogenomes of plants differ from those of animals in several aspects. First, plant mitogenomes comprise hundreds of thousands to millions of nucleotides ([45]), while those of animals are tens of thousands of bases. Second, the mitogenomes of plants recombine frequently, making the structure different from each other ([45]). Lastly and notably, mitochondria can have a gene that is responsible for a phenotype that the individual cannot produce mature pollen, which is called cytoplasmic male sterility (CMS) ([47]). Because of this unique effect on fertility, plant mitogenomes should be studied independently.

The first complete plant mitogenome came from *Arabidopsis thaliana*, using traditional Sanger sequencing ([143]) three years before the nucleus DNA was assembled ([51]). Since the emergence of short-read sequencers, many studies have assembled plants' mitogenomes as circular contigs ([102]). Long-read sequencers released by PacBio or Oxford Nanopore Technology have accelerated this process ([129, 130, 148, 31, 66, 156, 30, 67, 107, 81, 162]). Consequently, many studies have determined the nucleotide sequences of plant mitogenomes, compared the resulting assemblies, and proposed evolutionary scenarios ([156, 28]).

Nonetheless, there is room for improvement in obtaining a more in-depth understanding of mitogenome assemblies. One area that requires further clarification is the phenomenon of "alternative reality" ([67]). Plant mitogenomes are not static but change dynamically, and nucleotide sequences can vary due to recombination among repetitive sequences ([62, 8, 153, 45]). In other words, the plant mitogenomes have minor structures in addition to the major structures, comprising so-called "multipartite structures."

Although this phenomenon was observed in the study that first published a plant mitogenome assembly ([143]), plant mitogenomes are usually represented

as "master circles," i.e., single circular lists of nucleotides. Assays to reveal these multipartite structures based on polymerase chain reaction (PCR) are also not very useful, as they may create artifacts of PCR-mediated recombination ([5]).

Importantly, Kozik *et al.* recently presented a pioneering work that accounted for this insufficient representation ([67]). They first constructed an initial assembly to represent the major structure of the samples, then split the assembly into short "bulding blocks" of the genome, and "tiled" the long reads by these building blocks to investigate the multipartite structures.

Here, to further improve their work, I present the assembly of each strain of a sample by `JTK`. Comparison between strains showed massive recombinations and low substitutions between strains compared to the genomic DNA. Also, these assemblies provided the major structure for each strain and revealed the minor structures by the properly aligned long-reads.

3.2 Materials and methods

3.2.1 Datasets and software

I used the latest assembly (GenBank ID: BK010421.1) and the annotation (`GCF_000001735.4_TAIR10.1_genomic.gff`) as the reference.

I used nine datasets on *A.thaliana* from three sources. First, I downloaded seven WGS datasets of *A. thaliana* from a previous study ([57]). These strains were Kyoto, Sha, C24, An-1, L*er*, Eri-1, and Cvi-0 strains (SRA Accession numbers:ERR3415817-ERR3415831). I also used one whole-genome sequencing (WGS) dataset of *A. thaliana* L*er* strains available at PacBio's official repository (https://downloads.pacbcloud.com/public/SequelData/ArabidopsisDemoData/). Hereafter, I refer to this dataset as L*er*(2), as this is the second dataset on the L*er* strain. Additionally, as the current reference genome is the mitogenomes of the Col-0 strain, I sequenced this strain anew as a control dataset (SRA Accession numbers: DRR234977). These datasets were all CLR datasets sequenced by PacBio's Sequel systems.

I filtered out reads from nuclear genomes from these WGS datasets, taking into account the region from 3245K to 3511K bp in chromosome 2 of *A. thaliana* exhibits substantial similarity with the mitogenome. After filtering, I obtained 198 ± 50M bp (SD, n = 9) of long reads for each accession, presumably sequenced from the mitogenome. The average read lengths were 12.8±3.4K bp (SD, n = 9), and I achieved coverage of approximately 350 to 600. The error rate was approximately 13%, and the mismatch, deletion, and insertion rates were 3.3%, 3.6%, and 6.1%, respectively.

In the original publication ([87]), I used `Flye` to assemble the mitogenomes of these strains. It produced assemblies with acceptable qualities, but I could not analyze the sequence in detail because the assemblies produced by `Flye` at that time did not have sufficient base-level accuracy. In this thesis, I used `JTK` instead

of Flye and reconfirmed all the analyses in the original study. As shown in the Result section, JTK produced high-quality assemblies, and I could carry out additional analysis to study the relationship between mutations in the base level and recombinations in the structure level.

In the downstream analysis, I mainly used LAST ([61, 46]) to align the assemblies. For annotation, I used Liftoff ([132]) and minimap2 ([74]). For visualization, I used Gfaviz ([42]), AliTV ([6]), and Gepard ([68]). The other scripts used in the analysis are mainly written in Rust language and available at https://mlab.cb.k.u-tokyo.ac.jp/~ban-m/dthesis/mitogenome.tar.gz. All analyses were carried out on Intel®Xeon Platinum 8280 (2.7 GHz) processors with 192GB RAM.

3.2.2 Genome assembly and detection of structural variations

JTK accepts parameters through a configuration file. I used the parameters listed (List 1) as default and changed the haploid_coverage to 600 in the case of the L*er*(2) and Col-0 datasets.

I also run canu ([64]) version 2.2 with genomeSize=360K -pacbio parameter and Flye ([63]) version 2.9-b1774 with --genome-size 360K --pacbio-raw parameter.

After the assembly, I extract the alignments with large insertions and deletions using in-house scripts from the alignments produced by JTK.

3.2.3 Genome annotation

I annotated the assemblies by liftoff with default parameters. I regarded 13-mers that occurred more than 15 times across the assemblies as repeats. Precisely, I first concatenated the reference genome and all assemblies of the nine datasets produced by JTK, recorded the occurrences of every 13-mers, and regarded all 13-mers that occurred more than 15 times as *repetitive* regions. The result section provides the rationale of the parameters, i.e., the length of the k-mers and the threshold of the occurrences.

3.2.4 Variant calling

I called variants of these assemblies by aligning the assemblies to the reference genome by LAST. Precisely, I used the following pipeline:

1. `last-train $REFERENCE_GENOME $ASSEMBLY > train.par`
2. `lastal -p train.par $REFERENCE_GENOME $ASSEMBLY > aln.maf`
3. `cat aln.maf | last-split | last-split -r`

```
input_file = "input.fa"
haploid_coverage = 350
out_dir = "./"
prefix = "temp"

## Tunable parameters.
region_size = "360K"
threads = 1
to_polish = true
resume = false
read_type = "CLR"
kmersize = 12
top_freq = 0.001
purge_copy_num = 8
min_span = 2
verbose = 2
seed = 19090432890
chunk_len = 2000
margin = 500
exclude = 0.8
min_count = 10
component_num = 1
compress_contig = 15
polish_window_size = 2000
min_llr = 1
supress_ari = 0.4
match_ari = 4.0
mismatch_ari = -1.0
required_count = 7
```

Listing 1: The default parameters for JTK in the assemblies of mitogenomes.

3.2.5 Recombination analysis

To calculate the number of recombinations between two assemblies or between an assembly and the reference, I used `LAST` to obtain one-to-one alignments by the same approach as in the varial calling.

3.3 Results

3.3.1 Comparison between `JTK` and other assemblers

`JTK` produced assemblies on the nine datasets, and six of them were simple circular contigs (Fig. 3.1 and Table 3.1). On the remaining three datasets, `JTK` produced two contigs. The sizes of the contigs were around 360 Kbp, consistent with the size of the reference genome. `JTK` took from about 30 minutes to 50 minutes, depending on the coverage of the datasets.

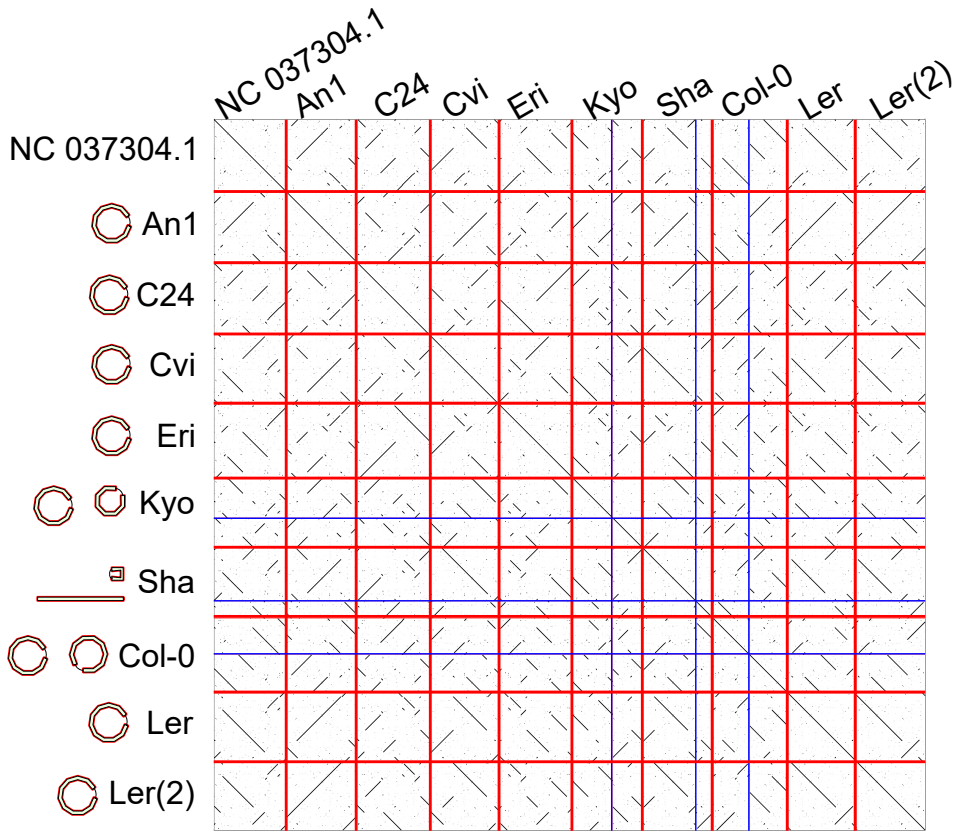

Figure 3.1: The assembly graphs on the strains and the dotplots between assemblies. The thick red lines and the thin blue lines represent the boundary of the assemblies between strains and the boundary between contigs in each strain, respectively.

Strain	Size (Kbp)	Contigs	Diffs	Genes	Coverage (Ref)	Coverage (Ctg)
An1	354	1	55	63	0.950	0.985
C24	370	1	75	65	0.992	0.985
Cvi	354	1	62	63	0.951	0.986
Eri	371	1	32	65	0.999	0.990
Kyo	356	2	34	63	0.943	0.974
Sha	360	2	130	62	0.910	0.928
Col-0	378	2	0	65	1.000	0.972
L*er*	348	1	30	63	0.936	0.989
L*er*(2)	351	1	27	63	0.935	0.980

Table 3.1: Summary of the assemblies produced by JTK. Size is the size of the contigs in Kbp. Diffs are the difference between the assemblies and the reference. The coverage (Ref) is the fraction of the bases in the reference covered by the assemblies. The coverage (Ctg) is the opposite.

One of the most important things in genome assembly is the correctness of the contigs. Although I did not have orthogonal datasets such as Illumina's short reads, I confirmed the quality of these assemblies by three approaches; checking the alignment between the reference genome, the gene annotations of the assemblies, and the coverage of the assemblies.

First, as the reference genome (NC_037304.1) is based on the Col-0 strain, I compare this reference sequence to the assembly based on the CLR from a col-0 strain. There were no mismatches, insertions, or deletions, i.e., the assembly was identical to the reference sequence in the alignments. Also, the assembly on the Col-0 dataset covered more than 99.99% bases of the reference sequence. These results suggest that the assembly was very accurate at the base-pair level.

Also, I compared two assemblies from the two L*er* datasets. By aligning these assemblies to the reference genome (NC_037340.1), I found 30 differences (two deletions, five insertions, and 23 mismatches) and 27 differences (two deletions and insertions and 23 mismatches) on these two assemblies, respectively. Out of these differences, 23 were shared by both assemblies of L*er* strains. Notably, all mismatches found in these assemblies were shared, suggesting that these mismatches were not consensus errors but single nucleotide polymorphisms among the strains. From these observations, I evaluated the quality value, $-10 \times \log_{10}(\text{probability of error})$, around 50, corresponding to one error out of 100Kbp.

In addition, all assemblies had 63 to 65 genes annotated by `Liftoff`, suggesting that there are no large insertions or deletions errors in the encoding region of the genes.

Next, I observed a coverage across strains assembled by JTK (Fig. 3.2). There were drops in the coverage at both ends of the contigs in some strains, presumably because the mitogenome has circular structures, making it hard to align the long read to both ends of the contigs. Except for these regions, I saw steady coverages

across all contigs in all strains.

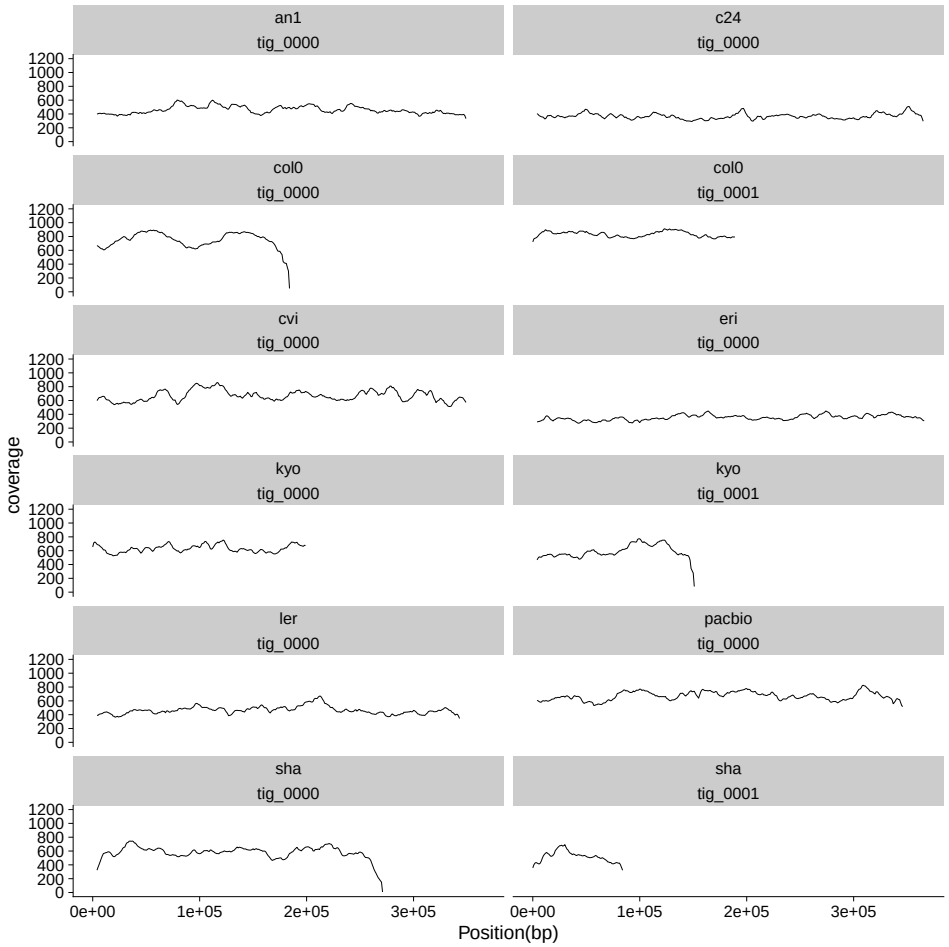

Figure 3.2: The coverage on the assemblies.

From these observations, I conclude that these assemblies accurately captured the major structures of the mitogenomes of the nine datasets accurately (QV > 50). To clarify the quality of these assemblies, I next briefly compared `JTK`'s assemblies to other software, i.e., `canu` and `Flye`.

Among many assemblers dedicated to PacBio's CLR datasets, I selected `canu` and `Flye` because these two assemblers **had been** most widely used [1].

`Canu` produced assemblies on the nine datasets (Table 3.2). Also, the gene annotations were highly consistent with those of `JTK`, suggesting that these two assemblies represented the same set of genes in the mitogenomes. In addition, the assemblies on each strain covered the reference sequence with a similar rate to the assemblies produced by `JTK`.

[1] By 2022, PacBio seemed to stop providing CLR sequencing kits, and there will be few datasets sequenced by CLR mode in the future.

Nonetheless, it produced fragmented assemblies. Specifically, it produced more than two contigs on all the datasets I tested. These contigs were also highly redundant. For example, the coverage rate of the Col-0 strain was around 60%, indicating that 40% of the contigs were redundant. Also, there were ten differences between the Col-0 strain and the reference assembly, which was higher than that of JTK.

Combining these results, I argue that Canu recovered the genes encoded in the mitogenomes. Still, the assemblies were highly redundant, and the base-level qualities were lower than JTK.

Strain	Size (Kbp)	Contigs	Diffs	Genes	Coverage (Ref)	Coverage (Ctg)
An1	438	4	60	63	0.950	0.957
C24	474	4	80	65	0.994	0.926
Cvi	423	3	57	63	0.950	0.925
Eri	497	6	38	65	0.999	0.805
Kyo	450	4	44	63	0.948	0.920
Sha	469	4	58	63	0.915	0.756
Col-0	810	7	10	65	1.000	0.616
L*er*	403	2	29	63	0.935	0.926
L*er*(2)	486	5	30	63	0.935	0.904

Table 3.2: Summary of the assemblies produced by Canu. See Table 3.1 for descriptions of the columns.

Flye produced assemblies on the nine datasets, usually similar to JTK in terms of both size, error rate, and coverages (Table 3.3). The error rates were similar to JTK, presumably because Fley and JTK used a similar approach to take a consensus. Precisely, Flye's consensus module is highly influenced by HGAP assembler's Quivar consensus module, which maximizes the likelihood of a pair-HMM ([24]). The only differences between JTK's and Flye's assemblies were the number of contigs in the C24 strain and 10% redundancy in the Col-0 strain. Thus, in terms of the assembly metrics I used, Flye was on par with JTK. Nonetheless, because JTK provided slightly better assemblies in the C24 and the Col-0 strain, I used the contigs produced by JTK in the downstream analysis.

In summary, JTK assembled the nine datasets into non-redundant, compact, and high-quality contigs. In the next section, I investigate the structural diversity and nucleotide differences using these assemblies.

3.3.2 Structural diversity and nucleotide differences among the nine strain of *A.thaliana*

The structure of plant mitogenomes frequently varies, presumably by recombinations mediated by repeats. This unique property has attracted many researchers,

Strain	Size (Kbp)	Contigs	Diffs	Genes	Coverage (Ref)	Coverage (Ctg)
An1	351	1	66	63	0.952	0.997
C24	366	4	76	65	0.975	0.977
Cvi	351	1	61	63	0.951	0.996
Eri	368	1	29	65	0.998	0.996
Kyo	350	2	32	63	0.948	0.995
Sha	339	1	69	62	0.917	0.992
Col-0	407	3	1	65	0.995	0.898
L*er*	345	1	27	63	0.935	0.995
L*er*(2)	345	2	29	63	0.935	0.995

Table 3.3: Summary of the assemblies produced by `Flye`. See Table 3.1) for descriptions of the columns.

and the structural diversities of soybean ([48]), tabacco ([147]), rice ([39]), and three strains of *A. thaliana* ([8]) have been investigated, to name a few.

I used the long reads datasets from nine strains of *A. thaliana* to investigate the structural and nucleotide diversity, i.e., recombinations and substitutions, in the mitogenomes.

Fig. 3.1 showed the overview of the assemblies obtained through `JTK`, and Fig. 3.3 and Fig. 3.4) represent the strain-to-strain comparison between assemblies.

Fig. 3.6 shows the recombinations between every two strains (mean = 7.22 and sd = 3.41). Notably, there was one large recombination between two datasets on the same L*er* strain, which was supported by inspecting the alignment at the recombination breakpoint (Fig. 3.7).

Regarding the presence or absence of genes in these assemblies, only three genes showed polymorphisms. Precisely, two tRNA sequences and one ATPase subunit were missing in some strains as follows:

- gene-DA397_mgt22 (TRNK.2) was absent from the Sha strain and present in the remaining eight datasets. This region encodes tRNA for lysines (`https://www.arabidopsis.org/servlets/TairObject?id=500229463&type=locus`).

- gene-DA397_mgt16 (TRNS.5) was present in C24, Col-0, and Eri strains and absent from the remaining six datasets. This region encodes tRNA for serines (`https://www.arabidopsis.org/servlets/TairObject?id=500229509&type=locus`).

- gene-DA397_mgp17 (ATP6-2) was also present in C24, Col-0, and Eri strains and absent from the remaining six datasets. This region encodes ATPase subunit 6 (`https://www.arabidopsis.org/servlets/TairObject?id=500229510&type=locus`).

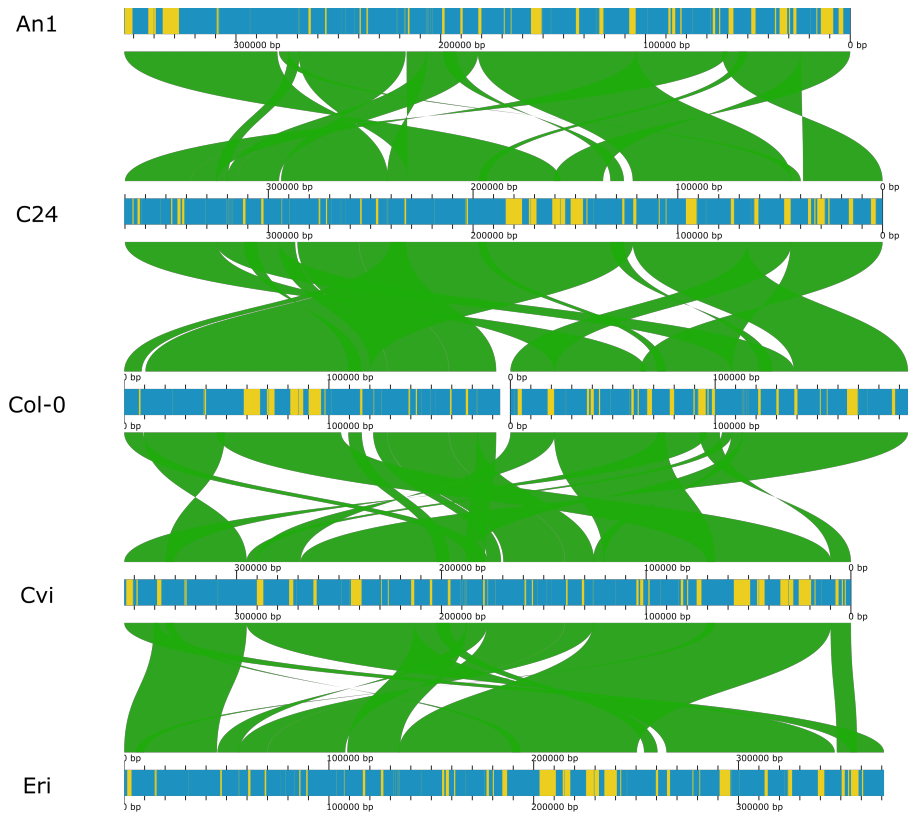

Figure 3.3: Strain-to-strain comparison of the mitogenome of *A.thaliana*. The blue and green bands represent the assembled sequences and the annotated genes. The green ribbons indicate the homologous regions between assemblies.

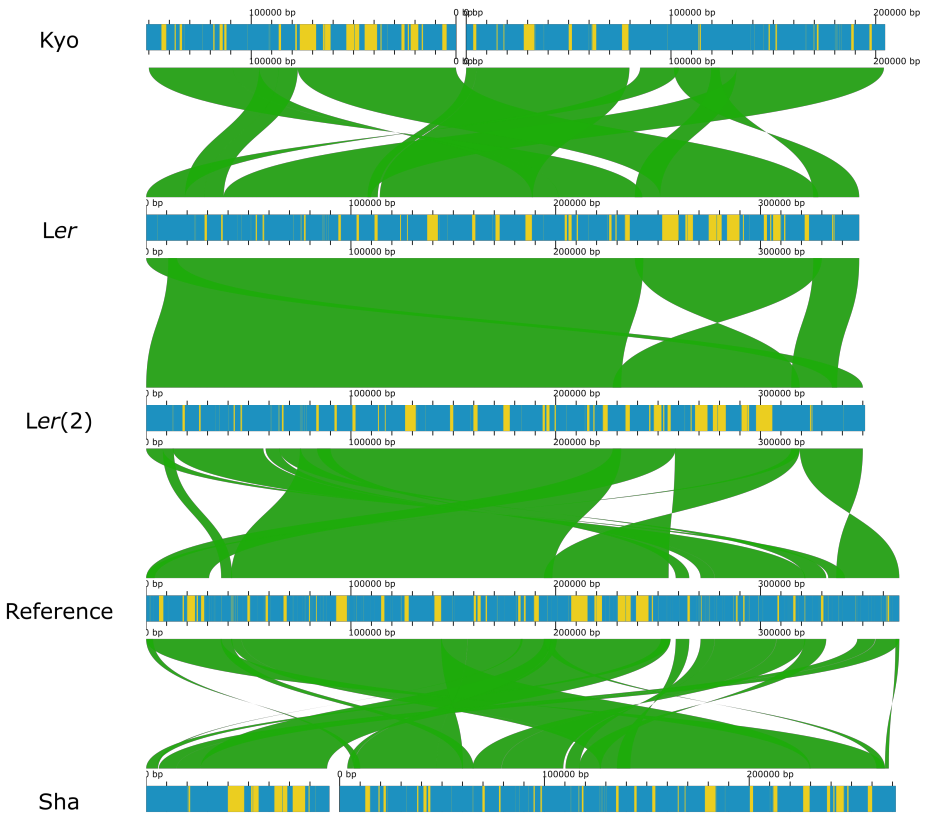

Figure 3.4: Strain-to-strain comparison of the mitogenome of *A.thaliana*. The blue and green bands represent the assembled sequences and the annotated genes. The green ribbons indicate the homologous regions between assemblies.

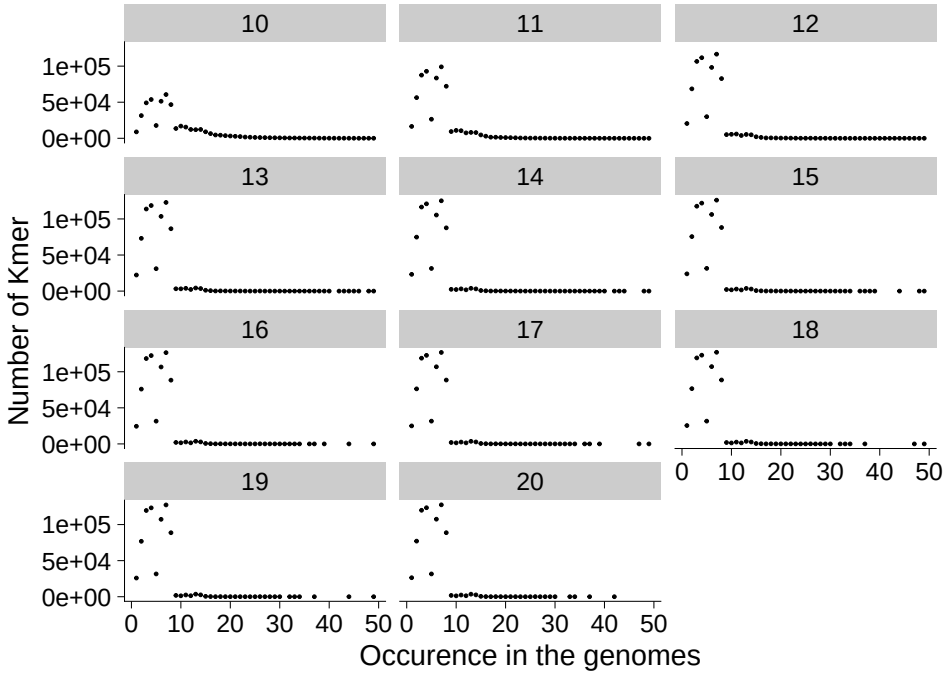

Figure 3.5: The k-mer histogram of the eight assemblies of mitogenomes of *A.thaliana*.

These recombinations among the strains of plants presumably occurred by repeat-mediated recombinations and nonhomologous end joining ([28]).

To confirm the recombination occurred mainly near repeats, I first annotated the repeats in the genome by a simple approach; I counted the k-mers in the eight assemblies and regarded k-mers occurred more than T times as "repetitive."[2] Then, I measured the distance from the recombinations to the nearest repetitive k-mers and compared the distance from a null distribution. To compute the null distribution, I sampled positions randomly from the assemblies and measured the distance to the nearest repeats. Although the values of k and T are highly arbitrary, the point is to use the null distribution (=randomly sampled genomic position) to statistically test the observed value.

Specifically, I computed the histogram of k-mers with varying k (Fig. 3.5) and regard $k = 13$-mers occurred more than $T = 15$ times as "repetitive" sequences. There were 156 recombinations among the assemblies, and the median distance from these recombinations to the nearest repeats was 44 bp. This value (44 bp) was significantly smaller than the positions sampled randomly from these assemblies (p-value $< 10^{-6}$).

The high-quality assemblies enabled me to compute the mutation rate be-

[2]Because I had two Col-0 assemblies and Ler assemblies, I exclude the reference sequence and the L*er*(2) strain to remove the redundancy.

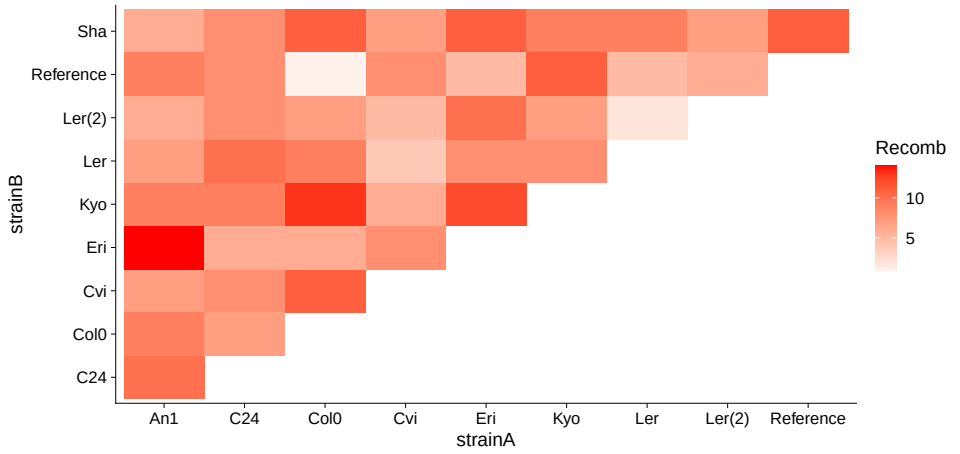

Figure 3.6: The heatmap indicating the number of recombinations between two strains of *A. thaliana*.

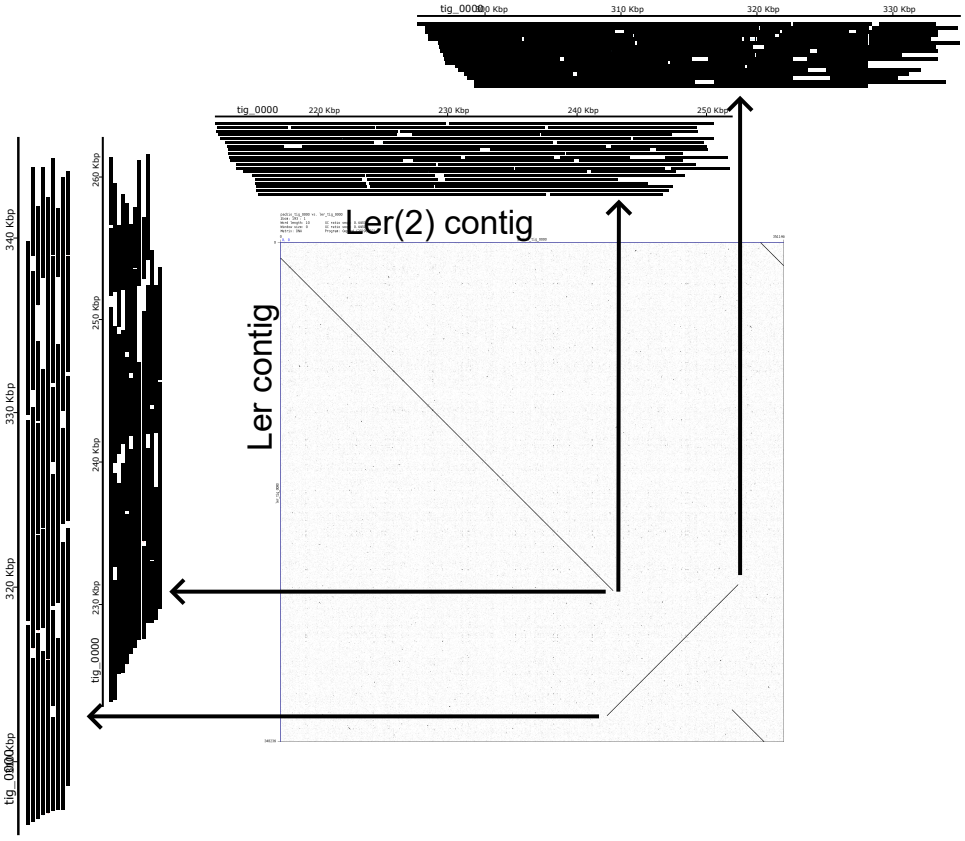

Figure 3.7: The recombination observed between two L*er* strains. I put the pile-ups of the alignments on the breakpoints, confirming the observed inversion. I subsampled 5%-10% of the alignments for visualization.

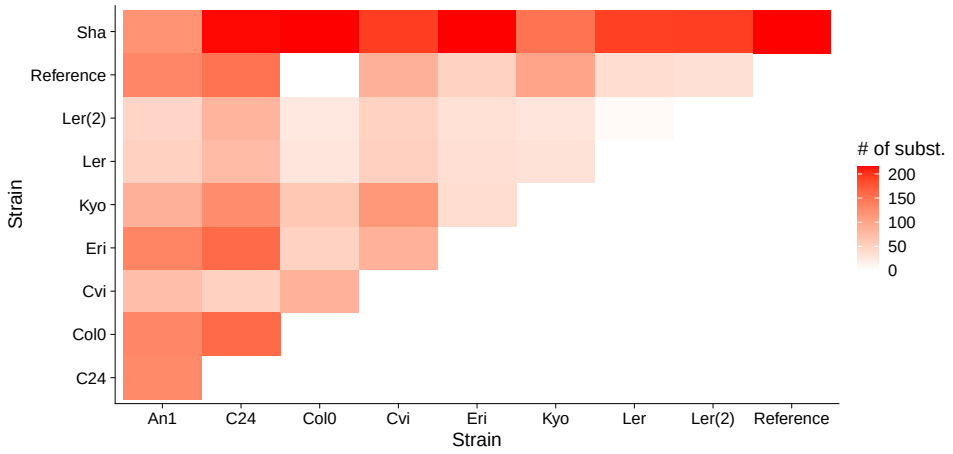

Figure 3.8: The number of substitutions in the nine strains and the reference sequence.

tween these assemblies.

Nonetheless, before analyzing the variations among the strain, it is essential to have reliable variations because even a small number of false variations can change the summary statistics significantly when the genome size is small. To this end, in the following analysis, I only focused on the substitution variations because PacBio's long reads are prone to introduce insertions and deletion errors, and there could be remaining consensus errors in the assemblies. Fig. 3.8 shows the number of substitutions between the assemblies.

Out of these observed substitutions, on the reference col-0 strain, I observed 129 SNPs. Three of these 129 SNPs were on the regions annotated as "exon", but they all represented non-coding RNAs. Precisely, the three SNPs were as follows:

1. The Sha strain had a substitution at 347,465bp (https://www.ncbi.nlm.nih.gov/gene/38088318).

2. Two L*er* stain had a substitution at 92,500bp (https://www.ncbi.nlm.nih.gov/gene/38088373).

3. Seven strains except Sha and Col-0 strain had a substitution at 333,207bp (https://www.ncbi.nlm.nih.gov/gene/?term=38088476).

As the previous study ([57]) assembled the genomic DNA with high contiguity, I could compare the recombination rate and substitution rate of the genomic DNA to the mitochondrial DNA. Specifically, I first normalized the number of SNPs and recombinations observed in the genomic DNA and the mitochondrial genomes by the size of the assemblies. The genomic and mitochondrial DNA showed clear separation (Fig. 3.9). Namely, mitochondrial DNA showed lower substitution rates and higher recombination rates compared to genomic DNA.

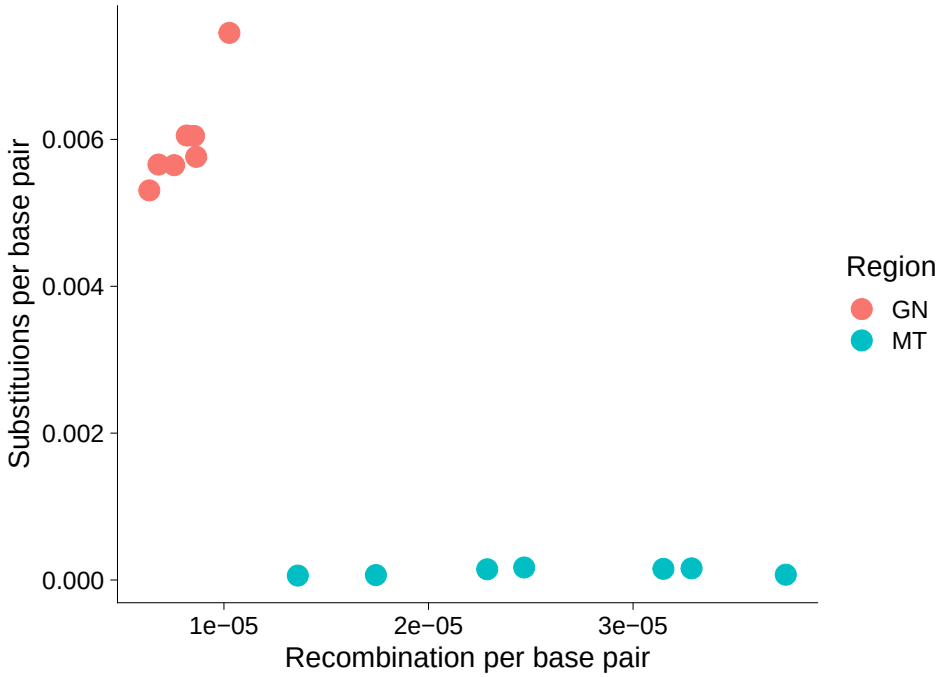

Figure 3.9: Scatter plot of the recombination and the substitution rate per base pair. The red points ("GN") represent the genomic DNA, and the blue ("MT" represent the mitochondrial DNA.)

3.3.3 Minor conformations in the mitogenomes provided by JTK's alignments

Mitochondrial genomes in plants recombine frequently and differ in structure from strain to strain. In addition, their structures differ even in an individual, which I call "minor conformations" in a sample. Although this structure-level heteroplasmy has been documented in many species and conditions, different studies have produced different estimations on the fraction of the minor conformation to the major one ([62, 154, 153, 162]).

JTK can produce not only the assembly but also the alignment between the assembly and the reads. Because all the data used in this study were PCR-free single-molecule sequencing, I could detect the minor conformations and the fraction of them quantitively.

Specifically, I selected alignments with large deletions and insertions in the genomes by in-house scripts (Fig. 3.10 to Fig. 3.17). Here, I put the assembled contigs at the top of the pileup and show the PacBio CLR reads below. The black rectangles showed the matched bases, and the blue lines indicated the deletions in the reads.

The PacBio SMRT sequencing can indeed produce chimeric reads, i.e., it can glue two different reads into one read, and an in-house scripts could include these chimeric reads in these figures. Also, it is true that these split alignments were artifacts due to repetitive sequences in the mitogenomes, and I need longer reads to span these repeats to rule out these artifacts.

Nonetheless, the probability of generating chimeric reads was presumably small, and some reads spanned the repetitive regions in the datasets. Thus, I argue that the structural variations supported by more than three reads were true minor conformations. The estimated fraction of these minor structures was as much as ~ 1% in the datasets.

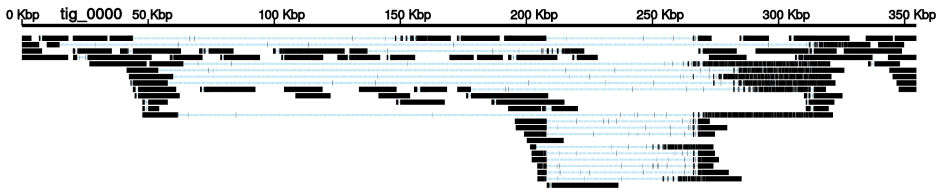

Figure 3.10: The pileup of the long reads of **An1** strain.

In addition to these minor conformations detected through large insertions in the alignments, I also detected alignments that stop at a certain position in the assemblies (Fig. 3.19 shows the L*er*(2) strain). Also, the reads were properly mapped in these alignments. In other words, there were no clipped region at both ends of these reads. Given that there were as many as 20,000 reads in this dataset

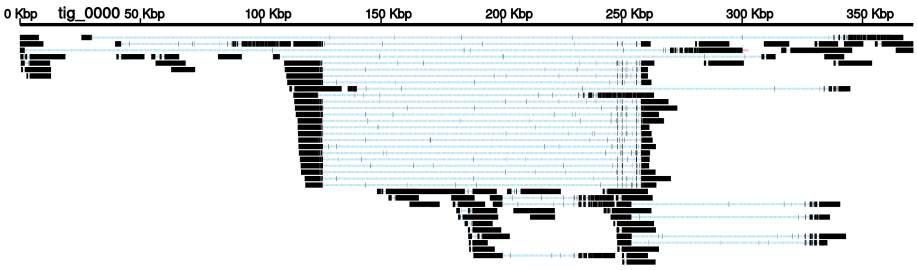

Figure 3.11: The pileup of the long reads of **Eri** strain.

Figure 3.12: The pileup of the long reads of **Col-0** strain.

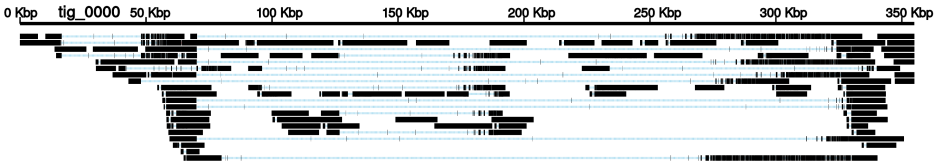

Figure 3.13: The pileup of the long reads of **Cvi** strain.

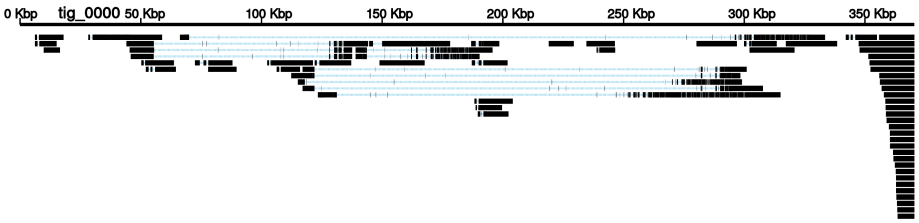

Figure 3.14: The pileup of the long reads of **C24** strain.

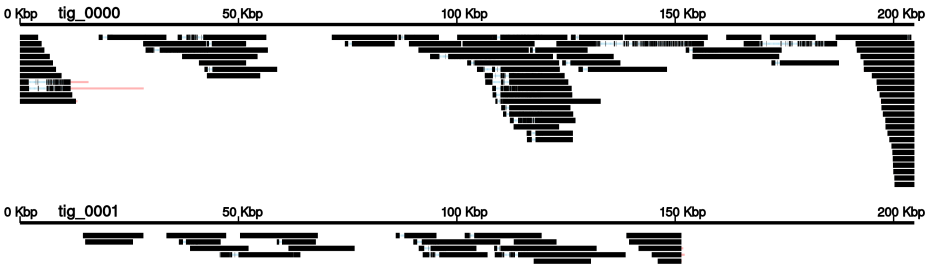

Figure 3.15: The pileup of the long reads of **Kyo** strain.

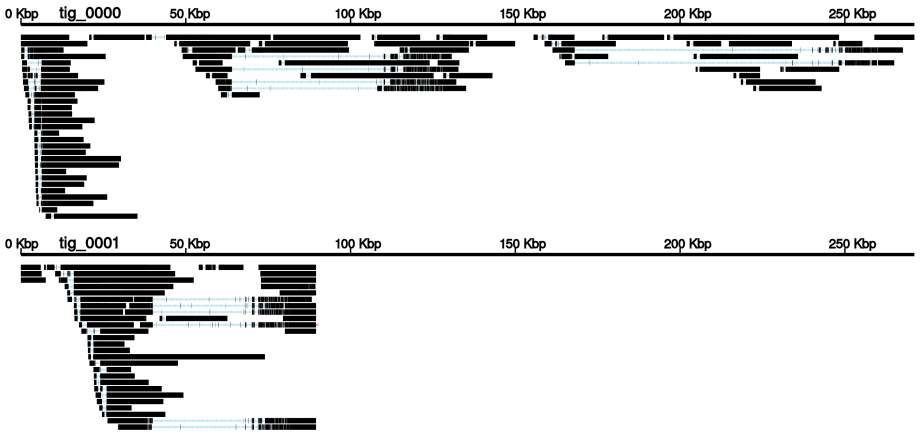

Figure 3.16: The pileup of the long reads of **Sha** strain.

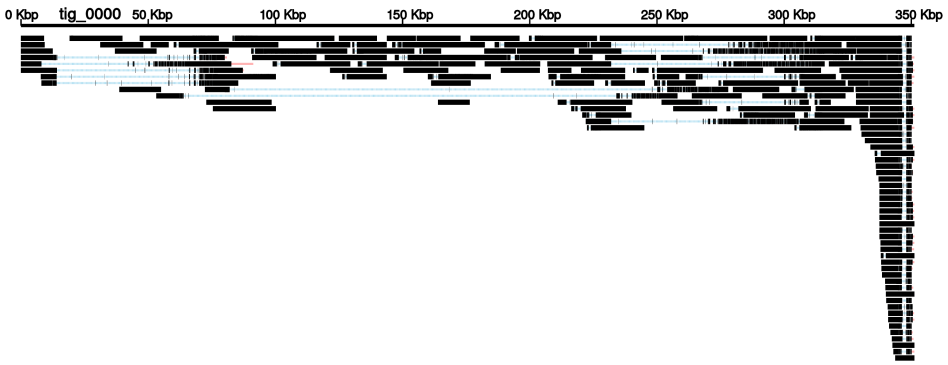

Figure 3.17: The pileup of the long reads **Ler** strain.

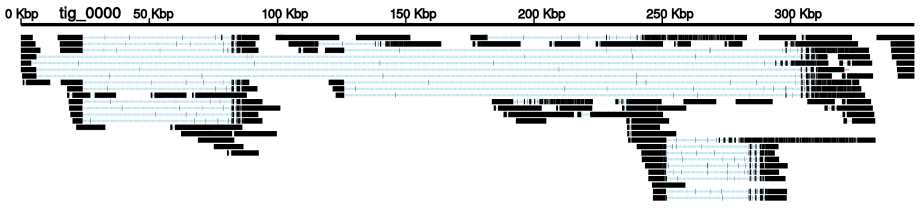

Figure 3.18: The pileup of the long reads **L*er*(2)** strain.

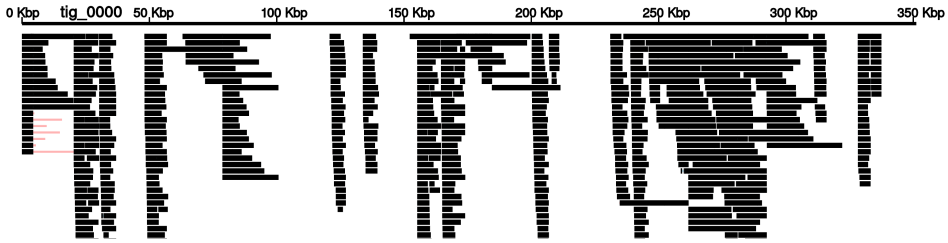

Figure 3.19: The putative linear structures in the Ler(2) datasets.

and the assembly size was 360Kbp, I should have observed $20000/360000 = 1/180$ reads on each position. Thus, statistically, it was unlikely (the p-values were below 0.001 in all cases) to see these correlated alignments on the pileup.

3.4 Discussions and Conclusions

3.4.1 Discussion

Plant mitochondrial genomes differ from those in animals in that they are large (several hundred thousand to millions of bases). In addition, in the plant mitochondria, the mitogenomes recombination even in an individual, creating minor conformations ([62]). Because of these factors, even though they are far smaller than the nucleus DNA, the complete assembly of the mitogenome using long noisy reads has not always been possible.

`JTK` produced more accurate and well-solved assemblies on the nine datasets than the other two assembles, `Canu` and `Flye`. Precisely, the assemblies on two L*er* strains were highly concordant at the base level, and the assemblies on the Col-0 strain exactly matched the reference (Col-0) assembly.

Of course, my methods and analyses have at least two limitations. First, because the read lengths were generally shorter than 15Kbp, it was not possible to detect whole-genome duplications (concatemers) of the mitogenome without variation ([106]). However, these possible recombination events do not introduce artifacts into the analyses. In other words, the recombinations observed in this study represent the lower limit of diversity.

Second, a small fraction of the reads in my datasets must be derived from nuclear DNA. To further clarify the minor conformations found in this study, it may be necessary to separately assemble the nuclear DNA and re-filter the mtDNA.

In the nucleus DNA, each strain contained as many as 8% unique sequences ([57]). In contrast, in this study, the reference sequence of the mitogenome of the Col-0 strain covered at least 97% of each assembly except the Sha strain. The Sha strain was the outlier and contained as many as 8% unique sequences in the assembly. Given that the assembly of the Col-0 strain covered 100% of the reference sequence, this high coverage rate was presumably not because the reference se-

quence was redundant. One explanation is that the mitogenome usually evolves slowly in plants, and the Sha strain was the most diverged strain in the datasets. This hypothesis is consistent with the fact that the base level difference of the Sha strain was higher than other strains (Fig. 3.8).

Previous studies have argued that recombinations occur by repeat-mediated recombinations ([136, 139], see [45] for a review). In *A. thaliana*, previous research proposed a structure of Col-0, C24, and L*er* strains ([28, 8]). Nonetheless, they used Illumina's short read sequencers to generate paired-end reads and estimated the recombinations through the alignments of these reads to the reference genome. JTK's assemblies revealed the mitogenomes of the eight strains, and the statistical analysis also supported that most of the recombinations were mediated by repetitive sequences in the genomes. By assembling the major structure *de novo*, I also found that the structure rearrangement among Col-0, C24, and L*er* strains was more complex than in the previous study. Further, the gene annotation analysis revealed that these recombinations rarely occurred in the encoding regions. One hypothesis is that whenever recombination occurs by a repetitive sequence in an encoding region, the resulting conformation is quickly eliminated from the cells.

The high frequency of recombination and the low rate of substitution are characteristics of plants' mitogenomes. A previous study estimated that the rate of substitution in the mitogenome is 16 times slower than in nucleus DNA ([33]). My analysis confirmed this tendency in diverse strains of *A. thaliana*.

In addition to the major structure, mitogenomes in plants recombine in an individual, creating minor structures or a heteroplasmic state ([153]). However, there has been room for improvement in a detailed analysis of the heteroplasmic state. Specifically, different studies proposed different fractions of these minor structures([62, 154, 153, 162]). By using long reads aligned to the assemblies by JTK, I estimated the fraction of these minor structures. Although I might miss rare conformations lower than 0.5% and repetitive sequences might create artificial minor conformations, I argue that the minor conformation comprises the genome as much as 5% in an individual in some strains. Also, I found putative linear structure from the alignments of the long reads, suggesting the presence of the linear structures in the mitogenome of *A. thaliana*, i.e., specific regions in the mitogenome are cropped in the cells. There are many possible explanations for these conformations. For example, they could be the result of incomplete replication or fragments of mitogenomes degraded by restriction enzymes (Dr.Frith, personal communication). However, this is pure speculation, and at the very least I need to analyze whether the DNA sequences at the boundaries of these liner structures have detectable features.

3.4.2 Conclusion

Plant mitochondrial genomes have unique characteristics such as large sizes compared to animals, recombinations by repetitive sequences, and minor structures.

As DNA sequencing becomes inexpensive and widely available, many "reference" mitogenomes of plants are now available.

Nonetheless, because the genome structures differ from strain to strain and individual to individual, it is not valid to use one reference sequence to analyze mitogenomes' landscape deeply.

Here, I used `JTK` to investigate the structural and nucleotide diversity of nine accessions of *A. thaliana*. Specifically, the *de novo* assemblies of these datasets revealed the complex structures among closely related strains, which short reads can not provide. Also, the alignments on these assemblies estimated the fraction of the minor structures in the datasets and putative linear structures.

"A genome is not a set of SNPs and SVs on a reference" - in this study, I represent the mitogenomes by a *de novo* assemblies and minor structures from them. This representation revealed the complex rearrangement among the closely related strains.

Nonetheless, the proposed approach is essentially based on comparing two assemblies or analyzing reads on an assembly. It would be a fascinating field of research to invent a method to analyze these assemblies through new data structures and algorithms, such as graph-based pan-genome analysis.

Chapter 4

The diploid assembly of major histocompatibility complex and leucocyte receptor complex in two human samples

Abstract

Each auto-chromosome has its homologous pairs in human somatic cells, comprising diploid genomes. As the differences between homologous chromosomes are known to be significant and medically necessary, separately assembling these sequences is essential. The sequence technologies and algorithms progress, and it is not a dream to generate diploid assembly.

There are two main approaches to producing a diploid assembly. (1) phasing reads based on variants and assembling the phased reads. (2) directly generate contigs by highly accurate reads (HiFi reads) and additional datasets such as ONT reads or Hi-C data. However, the former approach struggles to phase reads derived from large segmental duplications or structural variations. The latter approach is promising, but obtaining HiFi reads along with Hi-C or ONT reads would be a cost and time bottleneck when we want to have diploid assemblies on a population scale.

This section shows that the proposed software, `JTK`, produced diploid assemblies only from 60-fold ONT reads on target regions on two human samples.

Specifically, I assembled two medically essential regions, the major histocompatibility complex (MHC) and the leukocyte receptor complex (LRC) region. I also verified the performance of `JTK` on the subregion of chromosome 1. These assemblies showed superior contiguity and base quality compared to the HiFi-only and other ONT-only assemblies. Also, regarding the contiguity, they were on par with high-coverage HiFi with ONT or Hi-C datasets.

The gene annotations on these assemblies revealed the considerable haplotype divergence in the MHC and LRC region in the human genome, such as a segmental duplication in the MHC class III region. Also, although the two haplotypes of the LRC region in one sample were annotated as haplotype A, the major haplotype in Japanese populations, there was as much as 0.2% sequence divergence between haplotypes and a 3 Kbp expansion of a CT-rich region.

These results suggest we can assemble the human genome in a diploid resolution by long-noisy reads to analyze the haplotype variations deeply. In the near future, diploid assemblers like JTK will produce a lot of (more than 10,000) diploid assemblies in a human population. At that time, we will need a novel approach to analyze diploid genomes beyond SNP and SV level descriptions to elucidate the human genomes.

Code availability

The repository of JTK at https://github.com/ban-m/jtk includes scripts used in this chapter. The datasets used and the results on HG002 are available at https://doi.org/10.5281/zenodo.7192214. These files are also available upon request (banmasutani@gmail.com).

4.1 Introduction

In human genomes, each auto-chromosome has its homologous pair. The differences between them are known to be medically significant in cases such as the major histocompatibility complex (MHC) and the leukocyte receptor complex (LRC) ([50]).

For example, genome-wide association studies have frequently reported the correlations between variants on non-coding regions in the MHC region and common human diseases ([72]). Also, in bone marrow transplantation, non-major genes in these regions affect the inflammation risk ([91]). We need genotype and haplotype information in these cases due to high linkage disequilibrium and the prominent differences between haplotypes ([141]).

In addition, it is desirable to assemble diploid genomes at a population scale with minimal manual curation to understand how many variants we have missed ([54]). For example, a recent study sequenced 50 parent-child trios from the Danish population by short-read sequencers, statistically phased 100 MHC haplotypes, and found 357 novel structural variants (SVs) larger than 50 bp ([55]). These assemblies also found that the minor allele frequencies in the MHC region were higher than in the other regions in the genome, suggesting that the linked selection shelters polymorphisms in the MHC region. However, the authors build these assemblies with short-read sequencers, and the MHC and LRC regions contain many repetitive regions. As a result, the assembly pipeline sometimes failed to produce a fully phased assembly, and they could use only 50 haplotypes out of

200 sequenced haplotypes (= four haplotypes per one trio).

To further investigate the diversity among a population, we need an approach for the accurate diploid genomes assembly. Because the ONT's and PacBio's sequencers provide reads at least 100 times longer than the short reads, these reads can overcome the repetitive regions in the MHC and LRC regions where short reads can not resolve. Thus, it is promising to use these long-read sequencers to produce accurate, reliable, and completely phased assemblies of the MHC and LRC regions.

To this end, combining ONT's and PacBio's HiFi reads has been the de facto standard because they complement each other in lengths and error rates. Precisely, while ONT's sequencers provide reads even longer than 1 Mbp with a moderate error rate ($\sim$ 5%), PacBio's sequencers provide reads as long as 25 Kbp with very high accuracy ($<$ 0.1 %).

Combining these two reads and 10X linked reads, Chin *et al.* took below four steps to assemble the MHC region in a human sample named HG002 ([25]).

- They aligned all the reads to the reference genome (hs37d5).
- They located SNVs on the reference by using these alignments. Here, HiFi reads were useful for calling variants accurately.
- They phased SNVs using long reads and 10X linked reads and haplotyped the HiFi reads into parental or maternal haplotypes. ONT reads were useful to phase two variants separated by a very long homozygous region, i.e., a region without any variants.
- They assembled HiFi reads on each haplotype. Because of their high accuracy, the assembled contigs also showed high accuracy at the base level.

Although they could produce the diploid genomes and find heterozygous variants between haplotypes, their method had two drawbacks. First, their method depended on the reference assembly to find variants to be phased, which might cause unreliable variants or reference biases. Second, their method required high-coverage datasets from PacBio, ONT, and 10X linked reads. Precisely, they used 58-fold ONT reads, 34-fold HiFi reads, and 84-fold 10X linked reads. This requirement would be a problem when we want to scale their process to a broader sample, i.e., a human population. [1]

To solve the first issue, `DipAsm`([40]) first constructed a draft (haploid) sequence from the HiFi reads and then took a similar approach to Chin `et al.`

On the other hand, `HiFiASM`([22]) and `Verkko`([122]) used HiFi reads to construct diploid genomes and phase remaining homozygous regions by ONT reads or other datasets such as Hi-C.

[1] Also, 10X linked reads for a genomic sequence are no longer available in 2022 due to a patent issue.

Nonetheless, existing assembly strategies do require at least two types of datasets. For one thing, they need HiFi reads to construct highly accurate reads and variants. For the other, they require additional datasets such as ONT or Hi-C to infer the phasing information between two variants separated by a long homozygous region.

My proposed software, JTK, aims to create a diploid assembly of the hard-to-assemble region in the genome, such as MHC and KIR, with a single sequencing technique.

I showed that JTK produced diploid assemblies of the MHC, KIR, and subregion of chromosome 1 from 60-fold ONT reads on HG002 and a Japanese sample, which was impossible by a reference-based method and other ONT-only approaches. In addition, JTK's assemblies had high base-level concordance to the assemblies based on 100-fold HiFi reads, 85-fold ONT reads longer than 100 Kbp, and Hi-C reads, and the quality values (QV) were around 30.

I identified a highly diverged region on the MHC region in HG002 and observed that the set of genes inside the two haplotypes in that region was barely different. I also discovered that two haplotypes of the LRC region in the Japanese sample presumably belonged to the major haplotype in the Japanese population. Nonetheless, I found an expansion of a CT-rich repeat between the two haplotypes, suggesting significant variations between haplotypes.

In the future, diploid assemblers like JTK will provide many sequences of the previously thought hard-to-assemble regions. These assemblies will provide the landscape of variations in the regions and pave the way for the medical application of personal genomics.

4.2 Materials and Methods

4.2.1 Target regions and datasets

I run JTK on six real datasets with 60x coverages (Table 4.1). Then, I annotated these assemblies by `Liftoff` ([132]). The accession numbers of the HG002 and B080 (the Japanese sample) were SRR18363756-SRR18363760, and JGAS000580, respectively. I used the annotations on the CHM13-T2T version 2.

In the following sections, I refer to the LRC as leukocyte Ig-like receptor and killer-cell Ig-like receptor (LILR-KIR) region, as the LRC complex consists of two large domains, LILR and KIR.

In addition to the MHC and LILR-KIR regions, which are highly diverged, I also included the 10-15 Mbp region of chromosome 1, as this region contains a significant divergence in a population ([119]).

I obtained the genome-scale diploid assembly by `Verkko`([122]) from the HG002 data with 100x HiFi, 85x ONT UL(>100 Kbp), and Hi-C dataset to compute the QV. For an in-house sequence dataset from a Japanese sample, which is

referred to as B080 hereafter, `HiFiASM` ([22]) with 60x HiFi reads with Hi-C was used to generate the assembly.

Sample	Region	Total (Mbp)	N50 (Kbp)	Max len (Kbp)
HG002	MHC	320	45	230
HG002	LILR+KIR	70	46	150
HG002	Chr1sub	320	42	211
B080	MHC	320	57	720
B080	LILR+KIR	70	64	533
B080	Chr1sub	320	59	830

Table 4.1: Description of the dataset. MHC region is chr6:28,381,458-33,301,940, LILR+KIR is the chr19:57M-58M, and Chr1 sub is chr1:10,000,000-15,000,000 in the T2T-CHM13 reference version 2.0.

4.2.2 Software versions

In this study, I used the software programs listed in Table 4.2.

4.2.3 Filtering reads

To filter reads from the target regions, I used the following pipeline:

1. Align all the ONT reads to the T2T-CHM13 version 2 by minimap2.
   ```
   minimap2 -a --secondary=no -x map-ont $T2T $READS |
   samtools view -OBAM | samtools sort -OBAM
   ```
2. Index the bam file by `samtools index`
3. Filter out the reads.
   ```
   samtools view -OBAM $INDEXED_BAM $REGION | samtools fastq
   ```

4.2.4 Compared assemblers

Currently, HiFi-assemblers are the first choice to assemble genomes in diploid resolution. Thus, I used assemblies from `Verkko` and `HiFiASM` with only HiFi reads with around 60-fold coverage, which is as expensive as 60-fold ONT reads.

Also, I ran three different approaches for ONT reads. The first approach is traditional variant-calling followed by phasing with the latest software. I used `DeepVarinat` ([118]) and `CuteSV` ([56]) to call variants, `Longphase` ([77]) to phase them, and `Flye` ([63]) to assemble the reads from the haplotypes separately. Second and third, I used `PhaseBook` ([84]) and `PECAT` ([100]).

Software	HiFiASM	minimap2	rust
Version	0.16.1-r375	2.23-r1119-dirty	1.66.0-nightly
Software	AliTV	Flye	hifiasm
Version	1.0.6	2.9	0.16.1
Software	cuteSV	gepard	last
Version	1.0.3	1.40.0	1257
Software	Liftoff	longphase	nanosim
Version	1.6.3	1.2	3.1.0
Software	pecat	pepper_margin_deepvariant	phasebook
Version	1.2.12	r0.8	1.0.0

Table 4.2: Software programs and versions

4.2.5 Parameters of software programs

I used the following pipeline to use LongPhase and Flye to assemble diploig contigs:

1. Map the reads to the T2T-CHM13 reference by `minimap2 --MD -a -x map-ont | samtools sort -OBAM`

2. Call variants by DeepVariant by `run_pepper_margin_deepvariant call_variant --ont_r9_guppy5_sup`.

3. Call SVs by `cuteSV --report_readid --genotype --max_cluster_bias_INS 100 --diff_ratio_merging_INS 0.3 --max_cluster_bias_DEL 100 --diff_ratio_merging_DEL 0.3`, which was the recommended parameters provided by the authors.

4. Phase these SNVs and SVs by LongPhase by `longphase_linux-x64 phase --ont`. Alignments were tagged by `longphase_linux-x64 haplotag -s`.

5. Assemble the reads assigned to each haplotype with unphased reads by `flye --nano-raw` with appropriate `--genome-size` option.

JTK accepts parameters through a configuration file. I used the parameters listed (List 2) as default and changed the `region_size` to `"1M"` in the case of the LILR-KIR region.

```
## Required parameters.
input_file = "input.fa"
region_size = "5M"

## Tunable parameters.
read_type = "ONT"
out_dir = "./"
prefix = "temp"
threads = 1
to_polish = true
resume = false
kmersize = 12
top_freq = 0.001
purge_copy_num = 8
min_span = 2
verbose = 2
seed = 19090432890
chunk_len = 2000
margin = 500
exclude = 0.8
min_count = 10
component_num = 1
compress_contig = 15
polish_window_size = 2000
min_llr = 1
supress_ari = 0.4
match_ari = 4.0
mismatch_ari = -1.0
required_count = 7
```

Listing 2: The default parameters for JTK in the human diploid assembly.

4.3 Results

4.3.1 General comparison of assemblies

JTK fully resolved the targeted regions, MHC, LILR+KIR, and 10-15 Mbp in chromosome 1 from 60-fold ONT reads. They showed high concordance (QV$\sim$ 30) with baseline assemblies, which were assembled by HiFiASM or Verkko from high-coverage HiFi reads and ONT ultra-long reads or Hi-C reads. In the JTK assemblies, only less than 5% of errors were due to mismatch errors. Specifically, out of 16,297 errors in the assembly on the MHC region of the HG002, the number of deletions, insertions, and mismatches were 6056, 9531, and 710, respectively. There were differences between the JTK assembly and the Verkko assembly. Thus, these indels may not be due to the JTK consensus module but to the consensus errors in the HiFi reads.

Although Verkko and HiFiASM output complete assemblies for the five datasets with HiFi and ONT or Hi-C, they produced 18-133 fragmented contigs without ONT or Hi-C depending on the size of the assembled regions and the coverage of the reads. In addition, the quality values of the HiFi-only assemblies might be overestimated due to the following reason; The baseline assemblies used to compute QV were constructed from the same HiFi reads used for the HiFi-only assemblies. Thus, remaining errors in baseline assemblies might be shared with HiFi-only assemblies and do not affect the QVs of HiFi-only assemblies.

The phasing pipeline, LongPhase with Flye, produced unsolved assemblies for all datasets except LILR-KIR in the B080 sample. For example, in the chr1:10-15 Mbp region in HG002, the contigs were split at the SD-rich region (Fig. 4.3). The QVs of the contigs varied among the dataset, but they ranged from 20 to 30 in general. One explanation for the low QV is switching errors inside the contigs. In other words, like in the case of the synthetic dataset (Table 2.1 in Section 2.4), LongPhase incorrectly phased the variants and made chimeric haplotypes. Another explanation is that Flye's consensus module is not as accurate as JTK's, as Flye is mainly for fast chromosome-scale assembly while JTK is a megabase-scale assembler and tries to make the consensus as accurate as possible.

PECAT, an ONT-only assembler, output fragmented assemblies. For example, the LILR-KIR region in the HG002 was decomposed into five fragments of the partially phased contigs (Fig. 4.4). Similar to the phasing pipeline, the contigs terminated near repetitive sequences and overestimated the segmental duplications in the KIR region. In addition to this structural misassembly described, I noticed that PECAT produced redundant contigs in the LILR-KIR assembly (Fig. 4.4).

4.3.2 Analysis of the assemblies obtained by JTK

In the MHC region in HG002, I find a highly diverged region at the 1 Mbp of the assembled contigs upstream of the class II genes (Fig. 4.6). In this expanded region,

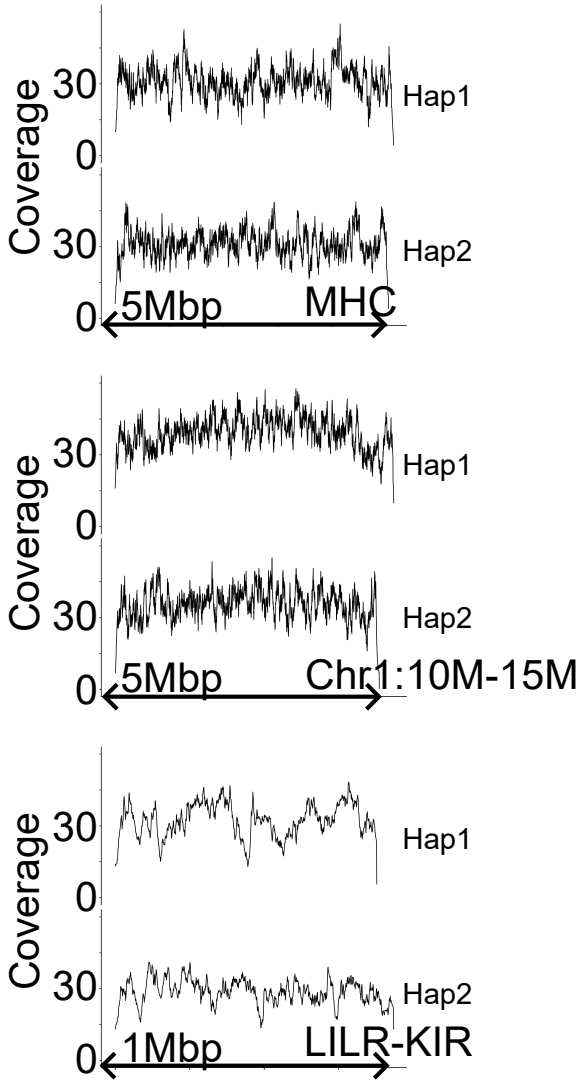

Figure 4.1: Coverage trajectories of JTK's contig in HG002.

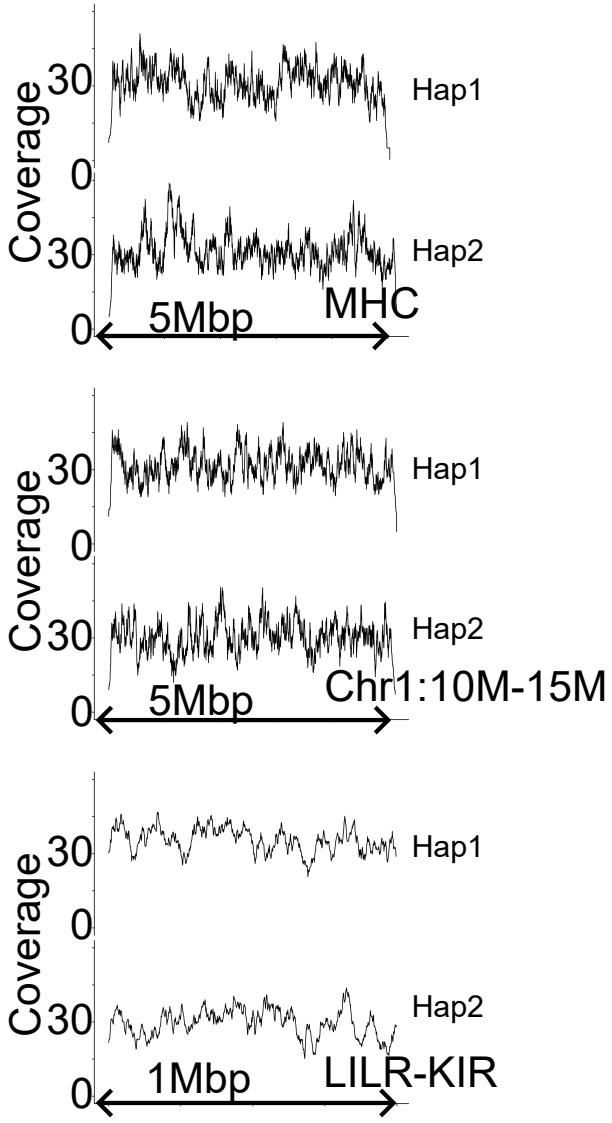

Figure 4.2: Coverage trajectories of JTK's contig in B080.

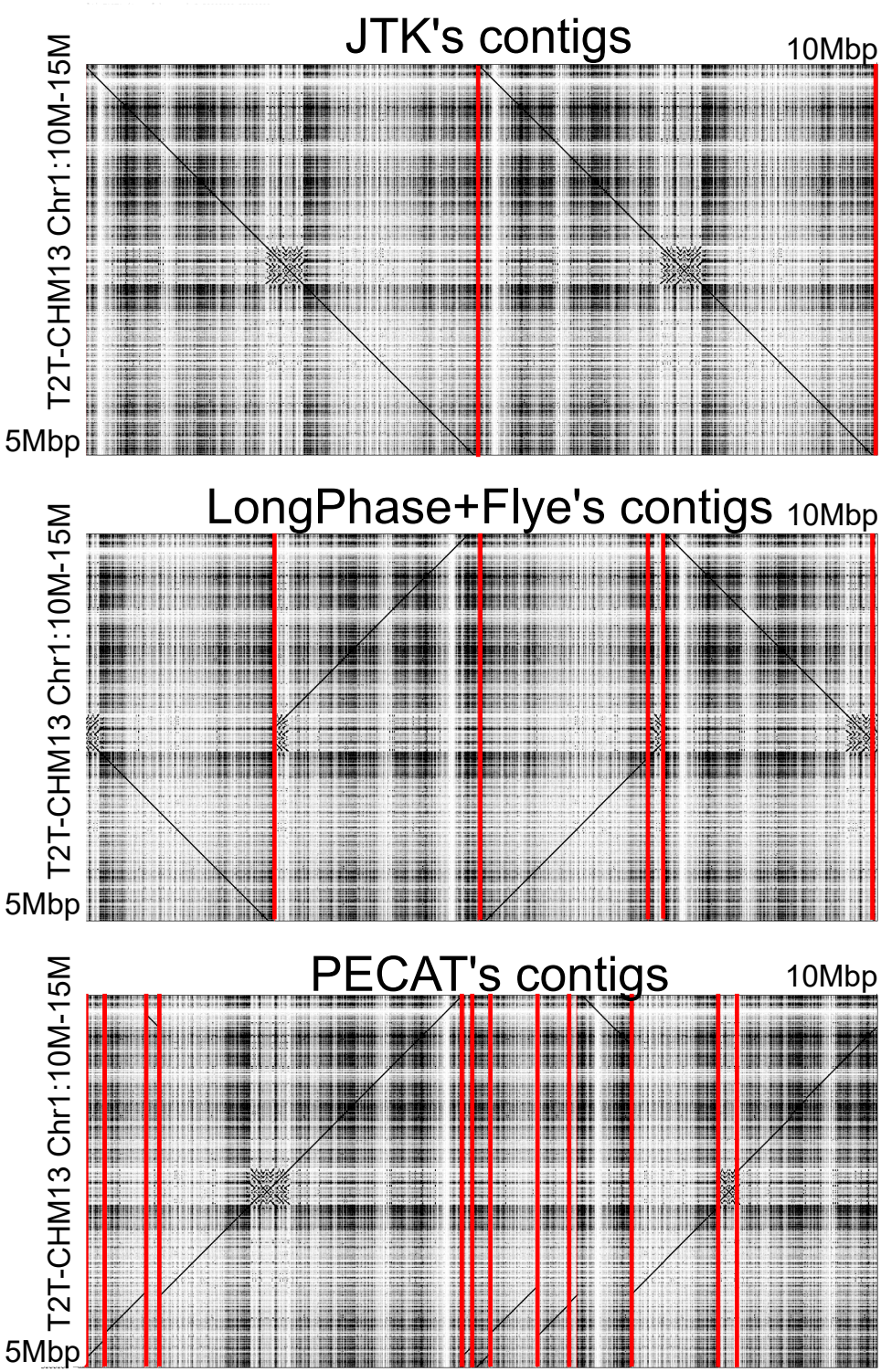

Figure 4.3: The dotplots between the assemblies and T2T-CHM13 chr1:10M-15M.

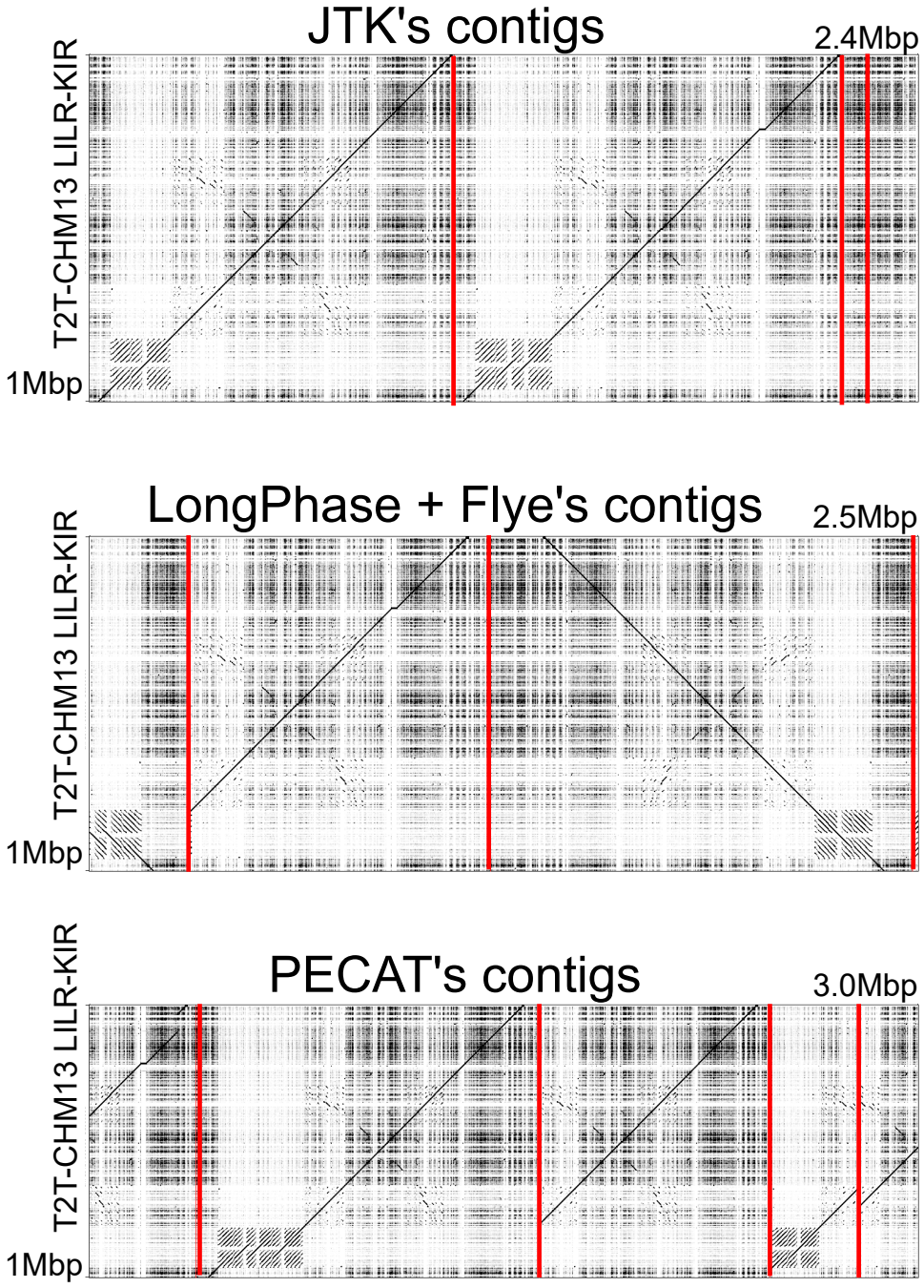

Figure 4.4: The dotplots between the assemblies and T2T-CHM13 KIR-LILR region.

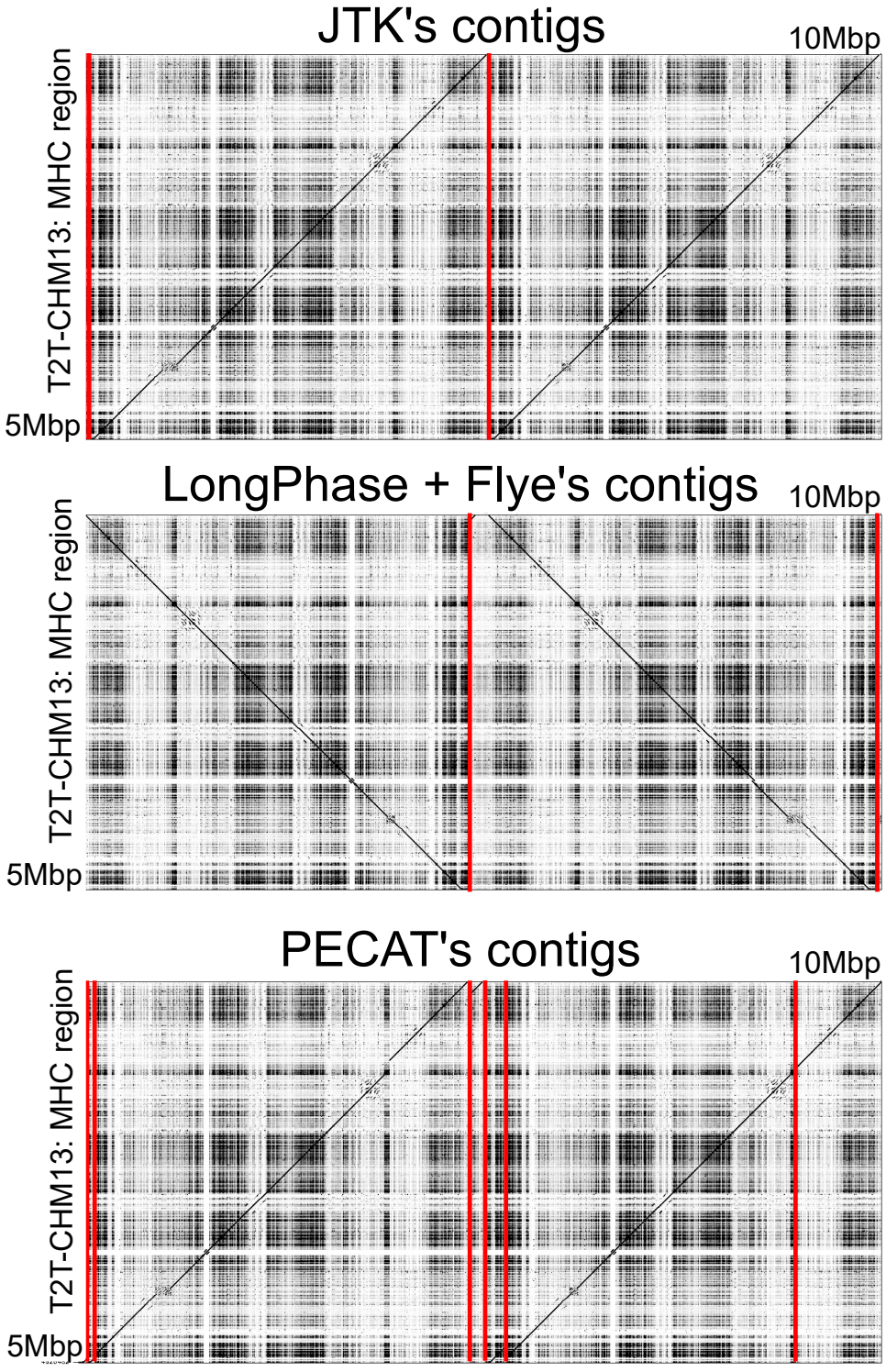

Figure 4.5: The dotplots between the assemblies and T2T-CHM13 MHC region.

one haplotype (haplotype 1) contained four genes, and the other (haplotype 2) had additional eight genes annotated by `Liftoff`. Specifically, the four shared genes are olfactory receptor genes (*OR12D1*, *OR11A1*, and *OR10C1*) and *AL645927*. Out of the eight additional genes, one was *MAS1L* (G protein-coupled receptor), and seven were described as pseudogenes or non-coding. I searched for *MAS1L* gene in haplotype1 to determine if this gene is missing or not in the other haplotype, and it was not missing in haplotype1; it had annotation in 35 Kbp upstream of this region, showing that the two haplotypes shared the same set of genes even when the order of the genes is shuffled in a diverged region.

In the MHC class III region in B080, I uncovered a copy number variation of 50 Kbp segmental duplication (two copies in one haplotype and four in the other, Fig. 4.7c), which I validated by a HiFi + Hi-C assembly. Both haplotypes had the same two annotated genes, *C4A* and *C4B*, and six annotated pseudogenes. These genes have different copies from individual to individual (https://www.genecards.org/cgi-bin/carddisp.pl?gene=C4B), demonstrating the capability of JTK for detecting the copy number variations of them.

In the LILR-KIR region in B080, I confirmed `JTK`'s assembly by observing that there were no coverage drops and by running `JTK` with a different seed for pseudo-random number generators to obtain the same structure. In contrast, `HiFiASM`'s assembly contained a 26 Kbp gap around 57.2 Mbp in chr19 of the T2T assembly. This gap contains two genes, *PRPF31* and *AC245052.4* (4.7b), which were annotated in the `JTK`'s contigs.

I found *KIR3DL3*, *KIR2DL3*, *KIR2DP1*, *KIR2DL1*, *KIR2DL4*, *KIR3DL1*, *KIR2DS4*, and *KIR2DL2* in both haplotypes in this order from the centromere to the telomere direction, suggesting both haplotypes were haplotype A, i.e., the major haplotype of this region. There were as many as 0.2% SNVs between two haplotypes (2574 substitutions out of 1.2 Mbp alignment), and I discovered a large ($\sim$ 3 Kbp) expansion of CT-rich sequence between the haplotypes, which was confirmed by alignment by the ONT reads (Fig. 4.8). In summary, the complete diploid assembly enables me to find significant divergences between the major haplotype in the KIR-LILR region.

4.4 Discussion and conclusion

4.4.1 Discussion

Comparison between `JTK` and other assemblers

For years, diploid assembly of erroneous reads seemed difficult because naive sequence identity between two reads would mix up SNVs between haplotypes and sequencing errors. `JTK` overcomes this issue and assembles genomic regions in diploid resolution from long-noisy reads.

The contigs produced by `JTK` were more contiguous than those from the HiFi-

Region	Software	# of contigs	Asm. size (Kbp)	QV
Chr1	JTK	2	10177	27.5
HG002	LongPhase	6	10324	24.7
	PECAT	14	10753	16.8
	Verkko	133	10845	35.3
LILR-KIR	JTK	2	2175	25.4
HG002	LongPhase	4	2484	21.2
	PECAT	5	3047	12.5
	Verkko	29	2416	35.0
MHC	JTK	2	10163	27.7
HG002	LongPhase	3	10542	22.9
	PECAT	7	10387	20.1
	Verkko	113	12044	36.8
Chr1	JTK	2	10242	29.1
B080	LongPhase	5	11116	28.5
	PECAT	4	9603	21.7
	HiFiASM	118	12284	40.5
LILR-KIR	JTK	2	2976	27.5
B080	LongPhase	3	2338	18.4
	PECAT	2	2302	26.2
	HiFiASM	18	2968	37.7
MHC	JTK	2	10111	30.8
B080	LongPhase	3	9502	19.3
	PECAT	3	10205	27.6
	HiFiASM	59	15874	55.6

Table 4.3: Comparison of the assemblies on six real datasets. The difference is the sum of the `NM` tag in `minimap2` after alignment to existing assemblies. Verkko means `Verkko` without Hi-C and ONT reads. HiFiASM means `HiFiASM` without Hi-C.

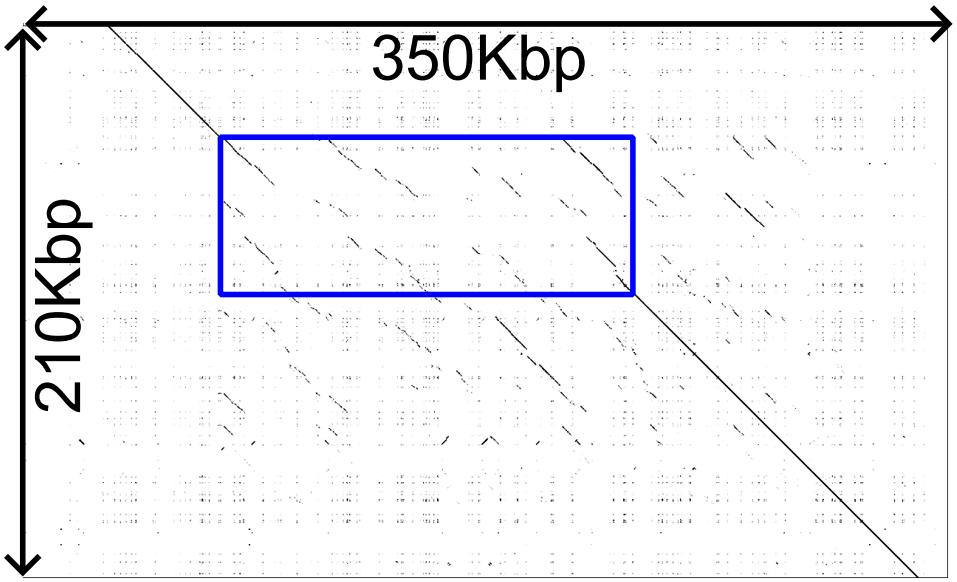

Figure 4.6: Dotplot of the highly variable region in the MHC region in HG002 assembled by JTK. The blue rectangle indicates the region I searched for annotations of genes.

only assembly from Verkko or HiFiASM. Specifically, for all real datasets, the HiFi-only assemblers output partially phased contigs while JTK phased the entire region. As the distance between adjacent variants can exceed 30 Kbp, it is impossible to phase these "SNV-deserts" by HiFi-reads. Therefore, the accuracy of the reads is not a sufficient condition for the diploid assembly, while the length of the reads can be.

Compared to phasing tools such as LongPhase, JTK can overcome errors caused by complex SVs and SDs. For example, the phasing-based method caused switching errors in the synthetic reads from the COX and PGF haplotypes, which were highly heterozygous. In contrast, JTK assembled these haplotypes correctly. The critical point might be using the chunks instead of the single reference sequence. If the heterozygosity is high, JTK can use different chunks to represent the region, which avoids the reference bias in the variant callers and the phasing tools.

PECAT is also an ONT-based assembler. However, in terms of contiguity and base accuracy, JTK generated better contigs. For example, in the 10-15 Mbp region in chromosome 1, JTK output two contigs with QV>27 fully resolving this region while PECAT produced 14 contigs with QV=16. One explanation is that while PECAT first corrects errors and then assembles the contigs, JTK uses uncorrected reads and variants on chunks. As error correction can result in overcorrection, PECAT can overlook essential SNVs and produce shorter phased blocks.

JTK has limitations and fails to produce phased assemblies on several regions,

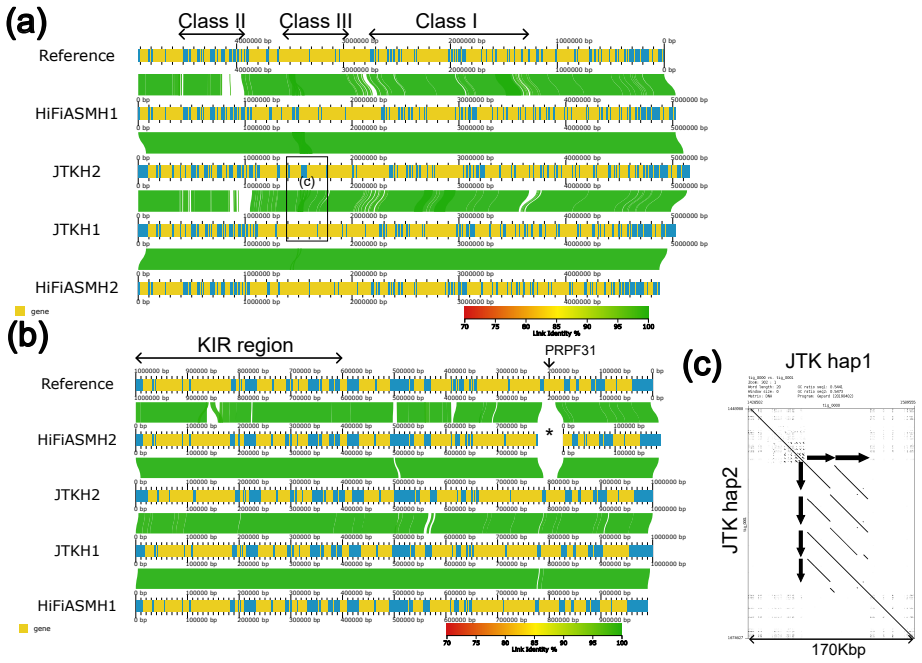

Figure 4.7: The comparison between the assemblies in the real datasets. Blue bars are contigs, yellow bands are annotations for genes, and green ribbons are the alignments. (a): The MHC region in HG002. I observed a large segmental duplication in the class II region (the black box), which is enlarged as (c). (b): The LILR-KIR in B080. The * mark indicates a large region missing from the `HiFiASM`'s contig. (c): An enlarged dotplot on the MHC region in B080. The arrows indicate a schematic illustration of the segmental duplications. I used `Gepard`([68]), `AliTV`([6]), `LAST`([61]), and `Liftoff`([132]).

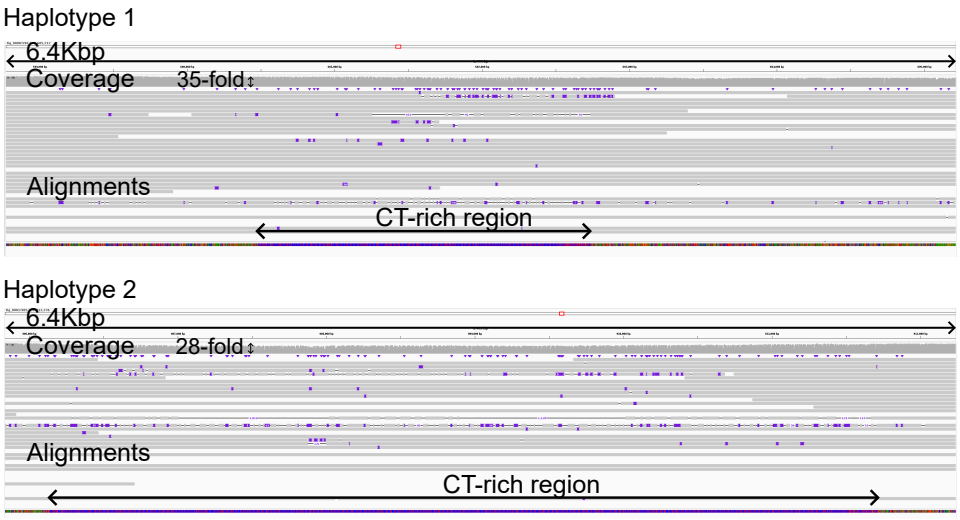

Figure 4.8: Alignments on the LILR-KIR region produced by `JTK`.

such as NOTCH2NL-NBPF (chr1:142-148 Mbp) and SMN (chr5:69-72 Mbp). These regions contain tens of segmental duplications longer than the 2 Kbp, complicating the intermediate assembly graph. I expect a more sophisticated algorithm to resolve repeats in assembly graphs would improve the assemblies in these regions.

The assemblies in the real datasets

First, it should be noted that the filtering step relies on the reference sequence, and thus I may miss some divergent or unique regions in the genome of the samples in general. However, as shown in Fig. 4.1 and Fig. 4.2, I argue that I did not miss any regions in this study.

When comparing two haplotype-resolved contigs, their differences must not be artifacts of the assembler. Speaking of the MHC region in both samples, I think the differences I observed were actual ones because the class II and III regions are highly variable, and complementary datasets (HiFi + Hi-C) produced the same structure ([55]).

In the HG002 sample, picked the upstream region of the MHC class II region, and in the B080 sample, I focused on one segmental duplication in the MHC class III region. In both cases, the gene prediction suggested that the gene contents were the same between the two haplotypes, only varying the non-coding regions or the copy number of *C4* genes. Although it should be analyzed rigorously and on a populational scale, I hypothesize from these observations that the haplotype differences around the MHC class II/III region have a small effect on the gene repertoire. In other words, the variations in the MHC region would not remove genes or insert a new gene but introduce SNVs or copy number variations.

Regarding the LILR-KIR region in the B080, there may be a switching error at the position where `HiFiASM` breaks contigs. Nonetheless, I support the correct-

ness of these haplotypes because the gene predictions were consistent with the reference, and JTK assembled nearly-identical haplotypes with different seeds for pseudorandom number generators.

On these assemblies, the annotation suggested that both haplotypes are the major haplotype in the Japanese population ([91]). Nonetheless, although there would be a small number of consensus errors remaining, I saw around 0.2% substitution between haplotypes, confirming the high divergence among the major haplotype. As the sequencing cost decreases, I expect the diploid assembly of the LILR-KIR would be widely available to describe the haplotype structure in fine detail, paving the way for medical applications of personal genomics.

4.4.2 Conclusion

Diploid assembly is an essential component to elucidate human genetic diversity. However, existing approaches either fail to assemble complex regions or require extensive data from different technologies. The presented software, JTK, assembled both simulated and real reads from a single long-read technology on regions previously thought to be hard to assemble. Combined with phasing tools for regions with low heterozygosity, regional diploid genome assemblers like JTK would be a cost-effective selection in the coming era of comparative pangenomics.

Chapter 5

Conclusion

The summer grass -
Trails of soldiers'
Dreams

Basho Matsuo

5.1 What does JTK solve?

I have devoted this thesis to explaining the method to reconstruct genomes from partial observations, i.e., *reads* from them. These reads have been much shorter than the genomes and contain errors. Also, genomes have repetitive sequences that make the assembly difficult. Due to these factors, our question about genome assembly is as follows:

On what condition can we assemble the genome from the reads?

Before the 2010s, assemblies were usually very fragmented and consisted of thousands of sequences per chromosome.

As time passed, the sequencing technologies improved to generate reads longer than 1 Mbp or with more than 99.9% accuracy. A lot of computer programs to assemble these reads have been developed (e.g., [22, 23, 105, 11]). These datasets and assemblers finally produced one of the long-standing dreams of human knowledge; we now have the gapless assembly of the human genome ([104]).

These genome assemblers share the same approach. They first use the highly accurate long reads, i.e., PacBio's HiFi reads, to assemble large parts of the genomes, leaving long homozygous regions or long exact repeats unsolved. Then, they use another dataset, such as ONT's ultra-long reads or Hi-C data, to assemble these remaining regions. The same approach is used for assembling more complex situations, such as assembling the homologous chromosomes separately, i.e., diploid assembly. In summary, they are verifying the following conjecture:

Having accurate reads and long reads is
a sufficient condition for genome assembly.

In this thesis, I tried to omit the first condition.

Having very long reads is a sufficient condition for genome assembly.

My software program, JTK, supports this conjecture by assembling mitogenomes in plants and several regions in the human genomes that were thought to be hard to assemble with erroneous long-reads.

I also note that another research validates this hypothesis by assembling the histone complex in *Drosophila melanogaster* with long but erroneous reads ([16]).

However, this proposition is not validated exhaustively even in a human genome; JTK could not assemble the SMN (chr5:69-72 Mbp) region in the human genome in diploid resolution. Because this region contains tens of exact repeats longer than 2 Kbp, it is a complex instance for JTK, which utilizes the chunks as long as 2 Kbps. Nonetheless, I observed that JTK could separate some of the repeats in this region. Also, the maximum length of the reads was as long as 500 Kbps, hinting that this failure was not due to the limitation of the ultra-long reads but the assembly algorithm after decomposing the chunks into their copies. I plan to implement the phased chunk graph in Section 2.3.7 into a string graph ([93]) via chunk-level alignments explained in Section 2.3.6.

5.2 A perturbation matrix is feature extraction

In JTK, I sample subsequences (*chunks*) from the input long-reads and separate these chunks into their copies in the genome. When I decompose the chunks into their copies, I use pair-hidden Markov models to find variants between copies and separate the reads R aligned to a chunk. The resulting clusters represent the copies of the chunk.

Specifically, for a read aligned to a chunk, I compute how the likelihood of the read changes when I introduce a variant into the chunk and denote it as the perturbation matrix P. Precisely, let $L(r|c)$ be the log-likelihood of the read $r \in R$ from the chunk c. [1] Also, let $\hat{c}_{(i,e)}$ be the chunk after modifying the i-th base of c by the edit operation e. For example, for $c = \texttt{AAGCT}$ and $(i, e) = (2, \mathrm{Del})$, $\hat{c}_{(i,e)}$ is AACT. Then, the perturbation matrix is defined as follows:

$$P_c[r][i][e] = L(r|\hat{c}_{(i,e)}) - L(r|c) \tag{5.1}$$

[1] Even though it is more natural to define the likelihood function as $L(c|r)$ because the sequence of the chunk is variable given a read, I use $L(r|c)$ to make the argument consistent with Section 2.

After selecting several informative variants $U = \{(i, e)\}$ as in Section 2.3.5, given the copy number K of the chunk, I partition the reads R into disjoint sets $R_1, \cdots, R_K$, maximizing the following objective function:

$$\sum_{k=1}^{K} \sum_{(i,e)\in U} \sum_{r\in R_k} \max(P_c[r][i][e], 0) \tag{5.2}$$

The algorithm that I use to compute P is not new. Jason Chin, one of the legendary bioinformaticians, developed the same algorithm in 2013 ([24]). Also, finding variant calling using the log-likelihood change is not new, either. `LongShot` proposed the same algorithm in 2019 ([36]). `Flye` implemented almost the same algorithm in their repeat-separation module ([63]).

The novel idea in `JTK` is that I convert a DNA string r into the fixed-length vector $P[r][\_][\_]$ representing the fold-change of the log-likelihoods. In contrast to the strings, the perturbation matrix quantitively expresses the variations by $P[r][i][e]$ and enables us to use any algorithms on real-valued fixed-length vectors.

Here, it is interesting to investigate the relationship between a pair-HMM and usual probabilistic models. Specifically, suppose we have an arbitrary probabilistic model M with the parameter θ, and the log-likelihood of an observation x on this model is $L(\theta|x)$.

Remember that I define the consensus c^* of the reads R as

$$c^* = \arg_c \max \sum_{r\in R} L(r|c) \tag{5.3}$$

This value corresponds to the maximum likelihood estimator of the model on a set of fixed observations X:

$$\theta^* = \arg_\theta \max \sum_{x\in X} L(\theta|X) \tag{5.4}$$

Also, given that the edit operation is the minimum change in the sequence, I can regard the Eq. (5.1) as a "derivative" of the likelihood, which is called the "score function" V in statistics:

$$V(\theta|x) = \lim_{h\to 0} \frac{L(\theta + h|x) - L(\theta|x)}{h} \tag{5.5}$$

Under these interpretations, it is natural to push these correspondences further (Table 5.1). For example, given the expectation operator $\mathbb{E}$, the Fisher information defined below plays a crucial role in statistics, such as in the Carmer-Rao inequality ([15]):

$$F(\theta|x) = \mathbb{E}\left[\left(\frac{d}{d\theta} \ln L(\theta|x) \right)^2 \right] \tag{5.6}$$

Is there a counterpart in the pHMM, and can we use it to derive useful information? It could be the Chapman-Robbins bond, which can be used to descrete variables such as DNA sequences (Dr. M. Frith, personal communication). It is exciting to see what algorithms we can derive from these correspondences.

Pair-HMM H	Probabilistic model M
Read r	Observation x
Chunk c	Parameters θ
Log-likelihood $L(r\|c)$	Log-likelihood $L(\theta\|x)$
$c^* = \arg\max_c \sum_r L(r\|c)$	$\theta^* = \arg\max_\theta \sum_x L(\theta\|x)$
Perturb. mat. P_{c^*}	Score func. $\frac{d}{d\theta} \ln L(\theta\|x)\|_{\theta=\theta^*}$
?	Fisher inf. $\mathbb{E}[(\frac{d}{d\theta} \ln L(\theta\|x))^2]\|_{\theta=\theta^*}$

Table 5.1: The putative correspondences between a pair-HMM and a probabilistic model.

5.3 Did genome assembly end?

Fifty years have passed since Sanger determined the sequence of the bacteriophage ϕX174 DNA ([127]). Since then, the algorithm of genome assembly and the sequencing technology have improved hand-in-hand. [2] In 2022, the researchers obtained the complete human genome constructed by combining highly accurate reads and long-range information among the human genome ([104]). Since then, more and more chromosome-scale assemblies have become available. In addition, assembling the diploid genomes is promising even though it still needs manual curations in complex regions such as segmental duplications or tandem repeats ([54]). Given that the performance of the DNA sequencers will keep improving, it is no longer reckless nor bold to ask a fundamental question. Is the genome assembly problem solved?

The answer is no for computer scientists. As stated at the very beginning of this thesis, a genome is the entire genetic information inherited from a **individual**, not species. If one wants to argue that their findings in the assembled genome are indeed those of the **species**, one needs to validate their assertion in the genomes of several samples of the same species. To this end, assembling a sample's genome as fast as possible and with as small data as possible is essential. Nonetheless, the current methods require more than 100-fold HiFi reads and ONT ultra-long reads, hindering the populational-scale genomics.

One promising approach is the so-called "resequencing" approach that leverages the fact that the genomes are similar except for several regions ([17]).

[2] To be fair, the technology always precedes the algorithms, but please be generous.

Specifically, I can assemble the regions that are presumably similar to the reference genome; all I need is to introduce the SNVs and SVs between the sample and the reference. If the genome is diploid, the problem becomes the haplotyping problem, such as a minimum error correction (MEC) problem ([79]). Even though the original publication proved the MEC problem is NP-hard(see [85] for another proof), there are many practical and efficient software programs for this problem.

On the regions where the divergence are high, I can assemble these regions by a dedicated approach, such as JTK proposed in this thesis. The MHC and LRC regions assembled in Chapter 4 are examples of these regions.

Given phasing software and JTK can run on a dataset as sharrow as 60-fold ONT reads, I am expecting this approach would solve the diploid problem at a populational scale.

For biologists, genome assembly is no longer an intensive international project. It has already become one of the projects that a single lab affords to do ([58]). Within a few years, asking a company to generate a gapless assembly of the genome of interest might even be commonplace. In this respect, the problem of genome assembly is fading away from the desk of biologists.

One possible direction we can take is to annotate these genomes and synthesize biological knowledge from them. Gene prediction ([128]), (single-cell) RNA-seq ([82, 137]), (single-cell) Hi-C ([76]), ATAC-seq ([158]), proteomics ([83]), and metabolomics are components of these annotations. [3] I expect that systems biology ([2]) plays a central role in interpreting these annotations in a unified manner.

The other approach is the opposite. This direction treats genomes as mere sequences and seeks to elucidate the similarities and evolutionary history of the massive number of genome sequences.

This approach is not new. For example, software programs such as Cactus and Progressive Cactus are tools for comparing a massive number of genomes ([7]). The method developed by Nakatani et al. also treats genomes purely as a large number of long sequences without annotation. By comparing these sentences, he discovered the evolutionary trail written in the "sentences" (i.e., genomes) ([96]). These two examples above did not require additional information such as RNA-seq or proteome. They also ignored annotations on the genome, and no distinction was made between coding regions and junk DNA, allowing for bias-free analysis.

I can push these approaches further. To take the most extreme example, I may even be able to remove the "species" annotation from the above analysis. Namely, a field could be formed where we collect many documents (i.e., genomes) without any species information and analyze them.

In the near future, every parent can determine the genome of their children, every farmer can determine the genome of their crops, and every researcher can determine the genome of their organism of interest. By analyzing these vast

[3] These cited papers are not reviews or the best papers in these fields. They are papers I enjoyed reading. [26] is also fun to read.

amounts of genomic data, I'm pretty sure that we will do something wrong – something that will and should be blamed by the people in the future. Nonetheless, as Evan Eichler said in his interview, as long as it is a legitimate study in light of one's own conscience, such a study is worth doing ([60]).

Bibliography

[1] Alberts, B. (2017). *Molecular biology of the cell*. WW Norton & Company. 14, 24

[2] Alon, U. (2006). *An introduction to systems biology: design principles of biological circuits*. Chapman and Hall/CRC. 151

[3] Altemose, N., Logsdon, G. A., Bzikadze, A. V., Sidhwani, P., Langley, S. A., Caldas, G. V., Hoyt, S. J., Uralsky, L., Ryabov, F. D., Shew, C. J., et al. (2022). Complete genomic and epigenetic maps of human centromeres. *Science*, 376(6588):eabl4178. 44

[4] Altshuler, D., Donnelly, P., and Consortium, T. I. H. (2005). A haplotype map of the human genome. *Nature*, 437(7063):1299–1320. 17

[5] Alverson, A. J., Rice, D. W., Dickinson, S., Barry, K., and Palmer, J. D. (2011). Origins and Recombination of the Bacterial-Sized Multichromosomal Mitochondrial Genome of Cucumber. *The Plant Cell*, 23(7):2499–2513. 105

[6] Ankenbrand, M. J., Hohlfeld, S., Hackl, T., and Förster, F. (2017). Alitv—interactive visualization of whole genome comparisons. *PeerJ Computer Science*, 3:e116. 106, 143

[7] Armstrong, J., Hickey, G., Diekhans, M., Fiddes, I. T., Novak, A. M., Deran, A., Fang, Q., Xie, D., Feng, S., Stiller, J., et al. (2020). Progressive cactus is a multiple-genome aligner for the thousand-genome era. *Nature*, 587(7833):246–251. 151

[8] Arrieta-Montiel, M. P., Shedge, V., Davila, J., Christensen, A. C., and Mackenzie, S. A. (2009). Diversity of the arabidopsis mitochondrial genome occurs via nuclear-controlled recombination activity. *Genetics*, 183(4):1261–1268. 104, 112, 124

[9] Arthur, D. and Vassilvitskii, S. (2006). k-means++: The advantages of careful seeding. Technical report, Stanford. 60

[10] Auton, A., Abecasis, G. R., Altshuler, D. M., Durbin, R. M., Bentley, D. R., Chakravarti, A., Clark, A. G., Donnelly, P., Eichler, E. E., Flicek, P., et al. (2015). A global reference for human genetic variation. *Nature*, 526(7571):68–74. 17

[11] Bankevich, A., Bzikadze, A. V., Kolmogorov, M., Antipov, D., and Pevzner, P. A. (2022). Multiplex de bruijn graphs enable genome assembly from long, high-fidelity reads. *Nature Biotechnology*, 40(7):1075–1081. 43, 147

[12] Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., et al. (2012). Spades: A new genome assembly algorithm and its applications to single-cell sequencing. *Journal of Computational Biology*, 19(5):455–477. PMID: 22506599. 36

[13] Bergström, A., McCarthy, S. A., Hui, R., Almarri, M. A., Ayub, Q., Danecek, P., Chen, Y., Felkel, S., Hallast, P., Kamm, J., et al. (2020). Insights into human genetic variation and population history from 929 diverse genomes. *Science*, 367(6484):eaay5012. 17

[14] Birney, E., Bateman, A., Clamp, M. E., and Hubbard, T. J. (2001). Mining the draft human genome. *Nature*, 409(6822):827–828. 24

[15] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer. 149

[16] Bongartz, P. and Schloissnig, S. (2018). Deep repeat resolution—the assembly of the Drosophila Histone Complex. *Nucleic Acids Research*, 47(3):e18–e18. 148

[17] Bourgeois, Y. X. C. and Warren, B. H. (2021). An overview of current population genomics methods for the analysis of whole-genome resequencing data in eukaryotes. *Molecular Ecology*, 30(23):6036–6071. 150

[18] Branton, D., Deamer, D. W., Marziali, A., Bayley, H., Benner, S. A., Butler, T., Di Ventra, M., Garaj, S., Hibbs, A., Huang, X., et al. (2008). The potential and challenges of nanopore sequencing. *Nature Biotechnology*, 26(10):1146–1153. 37

[19] Brown, T. A. (2018). *Genomes 4*. Garland science. 14

[20] Cavalli-Sforza, L., Wilson, A., Cantor, C., Cook-Deegan, R., and King, M.-C. (1991). Call for a worldwide survey of human genetic diversity: A vanishing opportunity for the human genome project. *Genomics*, 11(2):490–491. 17

[21] Check Hayden, E. (2014). Technology: The $1,000 genome. *Nature*, 507(7492):294–295. 37

[22] Cheng, H., Concepcion, G. T., Feng, X., Zhang, H., and Li, H. (2021). Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. *Nature Methods*, 18(2):170–175. 50, 55, 129, 131, 147

[23] Cheng, H., Jarvis, E. D., Fedrigo, O., Koepfli, K.-P., Urban, L., Gemmell, N. J., and Li, H. (2022). Haplotype-resolved assembly of diploid genomes without parental data. *Nature Biotechnology*. 55, 147

[24] Chin, C.-S., Alexander, D. H., Marks, P., Klammer, A. A., Drake, J., Heiner, C., Clum, A., Copeland, A., Huddleston, J., Eichler, E. E., et al. (2013). Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nature Methods*, 10(6):563–569. 73, 111, 149

[25] Chin, C.-S., Wagner, J., Zeng, Q., Garrison, E., Garg, S., Fungtammasan, A., Rautiainen, M., Aganezov, S., Kirsche, M., Zarate, S., et al. (2020). A diploid assembly-based benchmark for variants in the major histocompatibility complex. *Nature Communications*, 11(1):4794. 129

[26] Danchin, A., Ouzounis, C., Tokuyasu, T., and Zucker, J.-D. (2018). No wisdom in the crowd: genome annotation in the era of big data – current status and future prospects. *Microbial Biotechnology*, 11(4):588–605. 151

[27] Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., et al. (2011). The variant call format and VCFtools. *Bioinformatics*, 27(15):2156–2158. 48

[28] Davila, J. I., Arrieta-Montiel, M. P., Wamboldt, Y., Cao, J., Hagmann, J., Shedge, V., Xu, Y. Z., Weigel, D., and Mackenzie, S. A. (2011). Double-strand break repair processes drive evolution of the mitochondrial genome in Arabidopsis. *BMC Biology*. 104, 115, 124

[29] de la Rubia, I., Srivastava, A., Xue, W., Indi, J. A., Carbonell-Sala, S., Lagarde, J., Albà, M. M., and Eyras, E. (2022). Rattle: reference-free reconstruction and quantification of transcriptomes from nanopore sequencing. *Genome Biology*, 23(1):153. 52

[30] Diaz-Garcia, L., Rodriguez-Bonilla, L., Smith, T., and Zalapa, J. (2019). Pacbio sequencing reveals identical organelle genomes between american cranberry (Vaccinium macrocarpon ait.) and awild relative. *Genes*, 10(4):1–15. 104

[31] Dong, S., Zhao, C., Chen, F., Liu, Y., Zhang, S., Wu, H., Zhang, L., and Liu, Y. (2018). The complete mitochondrial genome of the early flowering plant Nymphaea colorata is highly repetitive with low recombination. *BMC Genomics*, 19(1):1–12. 104

[32] Driever, W. and Nüsslein-Volhard, C. (1988). A gradient of bicoid protein in drosophila embryos. *Cell*, 54(1):83–93. 14

[33] Drouin, G., Daoud, H., and Xia, J. (2008). Relative rates of synonymous substitutions in the mitochondrial, chloroplast and nuclear genomes of seed plants. *Molecular Phylogenetics and Evolution*, 49(3):827–831. 124

[34] Duffy, K., Arangundy-Franklin, S., and Holliger, P. (2020). Modified nucleic acids: replication, evolution, and next-generation therapeutics. *BMC Biology*, 18(1):112. 14

[35] Durbin, R. M., Altshuler, D., Abecasis, G. R., Bentley, D. R., Chakravarti, A., Clark, A. G., Collins, F. S., De La Vega, F. M., Donnelly, P., Egholm, M., et al. (2010). A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061–1073. 17, 46

[36] Edge, P. and Bansal, V. (2019). Longshot enables accurate variant calling in diploid genomes from single-molecule long read sequencing. *Nature Communications*, 10(1):4660. 54, 59, 149

[37] Falconer, E., Hills, M., Naumann, U., Poon, S. S. S., Chavez, E. A., Sanders, A. D., Zhao, Y., Hirst, M., and Lansdorp, P. M. (2012). Dna template strand sequencing of single-cells maps genomic rearrangements at high resolution. *Nature Methods*, 9(11):1107–1112. 71

[38] Frazer, K. A., Ballinger, D. G., Cox, D. R., Hinds, D. A., Stuve, L. L., Gibbs, R. A., Belmont, J. W., Boudreau, A., Hardenbol, P., Leal, S. M., et al. (2007). A second generation human haplotype map of over 3.1 million snps. *Nature*, 449(7164):851–861. 17

[39] Fujii, S., Kazama, T., Yamada, M., and Toriyama, K. (2010). Discovery of global genomic re-organization based on comparison of two newly sequenced rice mitochondrial genomes with cytoplasmic male sterility-related genes. *BMC Genomics*, 11(1):209. 112

[40] Garg, S., Fungtammasan, A., Carroll, A., Chou, M., Schmitt, A., Zhou, X., Mac, S., Peluso, P., Hatas, E., Ghurye, J., et al. (2021). Chromosome-scale, haplotype-resolved assembly of human genomes. *Nature Biotechnology*, 39(3):309–312. 54, 129

[41] Gavis, E. R. and Lehmann, R. (1992). Localization of nanos rna controls embryonic polarity. *Cell*, 71(2):301–313. 14

[42] Gonnella, G., Niehus, N., and Kurtz, S. (2018). GfaViz: flexible and interactive visualization of GFA sequence graphs. *Bioinformatics*, 35(16):2853–2855. 96, 106

[43] Green, P. (1997). Against a whole-genomeshotgun. *Genome Research*, 7(5):410–417. 24

[44] Green, R. E., Krause, J., Briggs, A. W., Maricic, T., Stenzel, U., Kircher, M., Patterson, N., Li, H., Zhai, W., Fritz, M. H.-Y., et al. (2010). A draft sequence of the neandertal genome. *Science*, 328(5979):710–722. 18

[45] Gualberto, J. M. and Newton, K. J. (2017). Plant mitochondrial genomes: dynamics and mechanisms of mutation. *Annual Review of Plant Biology*, 68:225–252. 48, 104, 124

[46] Hamada, M., Ono, Y., Asai, K., and Frith, M. C. (2016). Training alignment parameters for arbitrary sequencers with LAST-TRAIN. *Bioinformatics*, 33(6):926–928. 106

[47] Hanson, M. (2002). Plant Mitochondrial Mutations And Male Sterility. *Annual Review of Genetics*, 25(1):461–486. 48, 104

[48] He, T., Ding, X., Zhang, H., Li, Y., Chen, L., Wang, T., Yang, L., Nie, Z., Song, Q., Gai, J., et al. (2021). Comparative analysis of mitochondrial genomes of soybean cytoplasmic male-sterile lines and their maintainer lines. *Functional & Integrative Genomics*, 21(1):43–57. 112

[49] Horsthemke, B. (2022). A critical appraisal of clinical epigenetics. *Clinical Epigenetics*, 14(1):95. 15

[50] Houwaart, T., Scholz, S., Pollock, N. R., Palmer, W. H., Kichula, K. M., Strelow, D., Le, D. B., Belick, D., Lautwein, T., Wachtmeister, T., et al. (2022). Complete sequences of six major histocompatibility complex haplotypes, including all the major mhc class ii structures. *bioRxiv*. 128

[51] Initiative, T. A. G. (2000). Analysis of the genome sequence of the flowering plant arabidopsis thaliana. *Nature*, 408(6814):796–815. 104

[52] Institute, N. H. G. R. (2022). Definition:genome. https://www.genome.gov/genetics-glossary/Genome. 13

[53] Jain, M., Koren, S., Miga, K. H., Quick, J., Rand, A. C., Sasani, T. A., Tyson, J. R., Beggs, A. D., Dilthey, A. T., Fiddes, I. T., et al. (2018). Nanopore sequencing and assembly of a human genome with ultra-long reads. *Nature Biotechnology*, 36(4):338–345. 41

[54] Jarvis, E. D., Formenti, G., Rhie, A., Guarracino, A., Yang, C., Wood, J., Tracey, A., Thibaud-Nissen, F., Vollger, M. R., Porubsky, D., et al. (2022). Automated assembly of high-quality diploid human reference genomes. *bioRxiv*. 128, 150

[55] Jensen, J. M., Villesen, P., Friborg, R. M., Consortium, T. D. P.-G., Mailund, T., Besenbacher, S., and Schierup, M. H. (2017). Assembly and analysis of 100 full mhc haplotypes from the danish population. *Genome Research*, 27(9):1597–1607. 128, 144

[56] Jiang, T., Liu, Y., Jiang, Y., Li, J., Gao, Y., Cui, Z., Liu, Y., Liu, B., and Wang, Y. (2020). Long-read-based human genomic structural variation detection with cutesv. *Genome Biology*, 21(1):189. 54, 131

[57] Jiao, W.-B. and Schneeberger, K. (2020). Chromosome-level assemblies of multiple Arabidopsis genomes reveal hotspots of rearrangements with altered evolutionary dynamics. *Nature Communications*, 11(1):1–10. 105, 117, 123

[58] Jung, H., Ventura, T., Chung, J. S., Kim, W.-J., Nam, B.-H., Kong, H. J., Kim, Y.-O., Jeon, M.-S., and Eyun, S.-i. (2020). Twelve quick steps for genome assembly and annotation in the classroom. *PLOS Computational Biology*, 16(11):1–25. 151

[59] Karczewski, K. J. et al. (2020). The mutational constraint spectrum quantified from variation in 141,456 humans. *Nature*, 581(7809):434–443. 48

[60] Khamsi, R. et al. (2022). A more-inclusive genome project aims to capture all of human diversity. *Nature*, 603(7901):378–381. 152

[61] Kiełbasa, S. M., Wan, R., Sato, K., Horton, P., and Frith, M. C. (2011). Adaptive seeds tame genomic sequence comparison. *Genome Research*, 21(3):487–493. 106, 143

[62] Kmiec, B., Woloszynska, M., and Janska, H. (2006). Heteroplasmy as a common state of mitochondrial genetic information in plants and animals. *Current genetics*, 50(3):149–159. 104, 119, 123, 124

[63] Kolmogorov, M., Yuan, J., Lin, Y., and Pevzner, P. A. (2019). Assembly of long, error-prone reads using repeat graphs. *Nature Biotechnology*, 37(5):540–546. 41, 50, 106, 131, 149

[64] Koren, S., Walenz, B. P., Berlin, K., Miller, J. R., Bergman, N. H., and Phillippy, A. M. (2017). Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome Research*. 40, 106

[65] Korlach, J., Marks, P. J., Cicero, R. L., Gray, J. J., Murphy, D. L., Roitman, D. B., Pham, T. T., Otto, G. A., Foquet, M., and Turner, S. W. (2008). Selective aluminum passivation for targeted immobilization of single dna polymerase molecules in zero-mode waveguide nanostructures. *Proceedings of the National Academy of Sciences*, 105(4):1176–1181. 37

[66] Kovar, L., Nageswara-Rao, M., Ortega-Rodriguez, S., Dugas, D. V., Straub, S., Cronn, R., Strickler, S. R., Hughes, C. E., Hanley, K. A., Rodriguez, D. N., et al. (2018). Pacbio-based mitochondrial genome assembly of leucaena trichandra (leguminosae) and an intrageneric assessment of mitochondrial rna editing. *Genome biology and evolution*, 10(9):2501–2517. 104

[67] Kozik, A., Rowan, B. A., Lavelle, D., Berke, L., Schranz, M. E., Michelmore, R. W., and Christensen, A. C. (2019). The alternative reality of plant mitochondrial dna: One ring does not rule them all. *PLoS genetics*, 15(8):e1008373. 104, 105

[68] Krumsiek, J., Arnold, R., and Rattei, T. (2007). Gepard: a rapid and sensitive tool for creating dotplots on genome scale. *Bioinformatics*, 23(8):1026–1028. 106, 143

[69] Lacal, I. and Ventura, R. (2018). Epigenetic inheritance: Concepts, mechanisms and perspectives. *Frontiers in Molecular Neuroscience*, 11. 15

[70] Lander, E. S., Linton, L. M., Birren, B., Nusbaum, C., Zody, M. C., Baldwin, J., Devon, K., Dewar, K., Doyle, M., FitzHugh, W., et al. (2001). Initial sequencing and analysis of the human genome. *Nature*, 409(6822):860–921. 24, 25, 43

[71] Lander, E. S. and Waterman, M. S. (1988). Genomic mapping by fingerprinting random clones: A mathematical analysis. *Genomics*, 2(3):231–239. 26, 27

[72] Lenz, T. L., Spirin, V., Jordan, D. M., and Sunyaev, S. R. (2016). Excess of Deleterious Mutations around HLA Genes Reveals Evolutionary Cost of Balancing Selection. *Molecular Biology and Evolution*, 33(10):2555–2564. 49, 128

[73] Li, H. (2016). Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110. 41

[74] Li, H. (2018). Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics*, 34(18):3094–3100. 95, 106

[75] Liao, W.-W., Asri, M., Ebler, J., Doerr, D., Haukness, M., Hickey, G., Lu, S., Lucas, J. K., Monlong, J., Abel, H. J., et al. (2022). A draft human pangenome reference. *bioRxiv*. 17

[76] Lieberman-Aiden, E., van Berkum, N. L., Williams, L., Imakaev, M., Ragoczy, T., Telling, A., Amit, I., Lajoie, B. R., Sabo, P. J., Dorschner, M. O., et al. (2009). Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293. 18, 42, 151

[77] Lin, J.-H., Chen, L.-C., Yu, S.-C., and Huang, Y.-T. (2022). LongPhase: an ultra-fast chromosome-scale phasing algorithm for small and large variants. *Bioinformatics*, 38(7):1816–1822. 54, 131

[78] Lin, Y., Yuan, J., Kolmogorov, M., Shen, M. W., Chaisson, M., and Pevzner, P. A. (2016). Assembly of long error-prone reads using de bruijn graphs. *Proceedings of the National Academy of Sciences*, 113(52):E8396–E8405. 41

[79] Lippert, R., Schwartz, R., Lancia, G., and Istrail, S. (2002). Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in bioinformatics*, 3(1):23–31. 151

[80] Lloyd, J. P. B. and Lister, R. (2022). Epigenome plasticity in plants. *Nature Reviews Genetics*, 23(1):55–68. 14

[81] Logacheva, M. D., Schelkunov, M. I., Fesenko, A. N., Kasianov, A. S., and Penin, A. A. (2020). Mitochondrial genome of fagopyrum esculentum and the genetic diversity of extranuclear genomes in buckwheat. *Plants*, 9(5):618. 104

[82] Luecken, M. D. and Theis, F. J. (2019). Current best practices in single-cell rna-seq analysis: a tutorial. *Molecular Systems Biology*, 15(6):e8746. 151

[83] Lundberg, E. and Borner, G. H. H. (2019). Spatial proteomics: a powerful discovery tool for cell biology. *Nature Reviews Molecular Cell Biology*, 20(5):285–302. 151

[84] Luo, X., Kang, X., and Schönhuth, A. (2021). phasebook: haplotype-aware de novo assembly of diploid genomes from long reads. *Genome Biology*, 22(1):299. 52, 53, 55, 97, 131

[85] Mäkinen, V., Belazzougui, D., Cunial, F., and Tomescu, A. I. (2015). *Genome-scale algorithm design*. Cambridge University Press. 151

[86] Mascher, M., Wicker, T., Jenkins, J., Plott, C., Lux, T., Koh, C. S., Ens, J., Gundlach, H., Boston, L. B., Tulpová, Z., et al. (2021). Long-read sequence assembly: a technical evaluation in barley. *The Plant Cell*, 33(6):1888–1906. 26

[87] Masutani, B., Arimura, S., and Morishita, S. (2021). Investigating the mitochondrial genomic landscape of arabidopsis thaliana by long-read sequencing. *PLOS Computational Biology*, 17(1):1–16. 105

[88] Mayer, K. F. X., Waugh, R., Langridge, P., Close, T. J., Wise, R. P., Graner, A., Matsumoto, T., Sato, K., Schulman, A., Muehlbauer, G. J., et al. (2012). A physical, genetic and functional sequence assembly of the barley genome. *Nature*, 491(7426):711–716. 26

[89] Meyer, M., Kircher, M., Gansauge, M.-T., Li, H., Racimo, F., Mallick, S., Schraiber, J. G., Jay, F., Prüfer, K., de Filippo, C., et al. (2012). A high-coverage genome sequence from an archaic denisovan individual. *Science*, 338(6104):222–226. 18

[90] Miki, Y., Swensen, J., Shattuck-Eidens, D., Futreal, P. A., Harshman, K., Tavtigian, S., Liu, Q., Cochran, C., Bennett, L. M., Ding, W., et al. (1994). A strong candidate for the breast and ovarian cancer susceptibility gene *BRCA1*. *Science*, 266(5182):66–71. 15

[91] Morishima, S., Ogawa, S., Matsubara, A., Kawase, T., Nannya, Y., Kashiwase, K., Satake, M., Saji, H., Inoko, H., Kato, S., et al. (2010). Impact of highly conserved HLA haplotype on acute graft-versus-host disease. *Blood*, 115(23):4664–4670. 128, 145

[92] Myers, E. W. (1995). Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290. PMID: 7497129. 22

[93] Myers, E. W. (2005). The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85. 32, 148

[94] Myers, E. W., Sutton, G. G., Delcher, A. L., Dew, I. M., Fasulo, D. P., Flanigan, M. J., Kravitz, S. A., Mobarry, C. M., Reinert, K. H. J., Remington, K. A., et al. (2000). A whole-genome assembly of *Drosophila*. *Science*, 287(5461):2196–2204. 33

[95] Nakamura, R., Motai, Y., Kumagai, M., Wike, C. L., Nishiyama, H., Nakatani, Y., Durand, N. C., Kondo, K., Kondo, T., Tsukahara, T., et al. (2021). Ctcf looping is established during gastrulation in medaka embryos. *Genome Research*. 19

[96] Nakatani, Y. and McLysaght, A. (2017). Genomes as documents of evolutionary history: a probabilistic macrosynteny model for the reconstruction of ancestral genomes. *Bioinformatics*, 33(14):i369–i378. 151

[97] National human genome research institute (2022a). Bacterial artificial chromosome. [Online; accessed 15-November-2022]. 23

[98] National human genome research institute (2022b). Yeast artificial chromosome. [Online; accessed 15-November-2022]. 23

[99] NatureEducation (2022). Definition:genome. https://www.nature.com/scitable/definition/genome-43/. 13

[100] Nie, F., Huang, N., Zhang, J., Ni, P., Wang, Z., Xiao, C.-L., Luo, F., and Wang, J. (2022). de novo diploid genome assembly using long noisy reads via haplotype-aware error correction and inconsistent overlap identification. *bioRxiv*. 53, 55, 131

[101] Noonan, J. P., Coop, G., Kudaravalli, S., Smith, D., Krause, J., Alessi, J., Chen, F., Platt, D., Pääbo, S., Pritchard, J. K., et al. (2006). Sequencing and analysis of neanderthal genomic dna. *Science*, 314(5802):1113–1118. 18

[102] Notsu, Y., Masood, S., Nishikawa, T., Kubo, N., Akiduki, G., Nakazono, M., Hirai, A., and Kadowaki, K. (2002). The complete sequence of the rice (Oryza sativa L.) mitochondrial genome: frequent DNA sequence acquisition and loss during the evolution of flowering plants. *Molecular Genetics and Genomics*, 268(4):434–445. 104

[103] Nowoshilow, S., Schloissnig, S., Fei, J.-F., Dahl, A., Pang, A. W. C., Pippel, M., Winkler, S., Hastie, A. R., Young, G., Roscito, J. G., et al. (2018). The axolotl genome and the evolution of key tissue formation regulators. *Nature*, 554(7690):50–55. 39

[104] Nurk, S., Koren, S., Rhie, A., Rautiainen, M., Bzikadze, A. V., Mikheenko, A., Vollger, M. R., Altemose, N., Uralsky, L., Gershman, A., et al. (2022). The complete sequence of a human genome. *Science*, 376(6588):44–53. 43, 44, 147, 150

[105] Nurk, S., Walenz, B. P., Rhie, A., Vollger, M. R., Logsdon, G. A., Grothe, R., Miga, K. H., Eichler, E. E., Phillippy, A. M., and Koren, S. (2020). Hicanu: accurate assembly of segmental duplications, satellites, and allelic variants from high-fidelity long reads. *Genome Research*, 30(9):1291–1305. 55, 147

[106] Oldenburg, D. J. and Bendich, A. J. (2001). Mitochondrial dna from the liverwort marchantia polymorpha: circularly permuted linear molecules, head-to-tail concatemers, and a 5′ protein11edited by n.-m. chua. *Journal of Molecular Biology*, 310(3):549–562. 123

[107] Omelchenko, D. O., Makarenko, M. S., Kasianov, A. S., Schelkunov, M. I., Logacheva, M. D., and Penin, A. A. (2020). Assembly and analysis of the complete mitochondrial genome of capsella bursa-pastoris. *Plants*, 9(4):469. 104

[108] OxfordLearnersDictionary (2022). Definition:genome. https://www.oxfordlearnersdictionaries.com/definition/english/genome. 13

[109] Ozaki, K., Ohnishi, Y., Iida, A., Sekine, A., Yamada, R., Tsunoda, T., Sato, H., Sato, H., Hori, M., Nakamura, Y., et al. (2002). Functional snps in the lymphotoxin-α gene that are associated with susceptibility to myocardial infarction. *Nature Genetics*, 32(4):650–654. 17, 44

[110] Parra, G., Bradnam, K., and Korf, I. (2007). CEGMA: a pipeline to accurately annotate core genes in eukaryotic genomes. *Bioinformatics*, 23(9):1061–1067. 39

[111] Parra, G., Bradnam, K., Ning, Z., Keane, T., and Korf, I. (2008). Assessing the gene space in draft genomes. *Nucleic Acids Research*, 37(1):289–297. 39

[112] Perez, M. F. and Lehner, B. (2019). Intergenerational and transgenerational epigenetic inheritance in animals. *Nature Cell Biology*, 21(2):143–151. 15

[113] Pevzner, P. A. (1989). l-tuple dna sequencing: Computer analysis. *Journal of Biomolecular Structure and Dynamics*, 7(1):63–73. PMID: 2684223. 34

[114] Pevzner, P. A., Tang, H., and Tesler, G. (2004). De novo repeat classification and fragment assembly. *Genome Research*, 14(9):1786–1796. 36

[115] Pevzner, P. A., Tang, H., and Waterman, M. S. (2001a). An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753. 35, 36

[116] Pevzner, P. A., Tang, H., and Waterman, M. S. (2001b). A new approach to fragment assembly in dna sequencing. In *Proceedings of the Fifth Annual International Conference on Computational Biology*, RECOMB '01, page 256–267, New York, NY, USA. Association for Computing Machinery. 34

[117] Piovesan, A., Pelleri, M. C., Antonaros, F., Strippoli, P., Caracausi, M., and Vitale, L. (2019). On the length, weight and gc content of the human genome. *BMC Research Notes*, 12(1):106. 18

[118] Poplin, R., Chang, P.-C., Alexander, D., Schwartz, S., Colthurst, T., Ku, A., Newburger, D., Dijamco, J., Nguyen, N., Afshar, P. T., et al. (2018). A universal snp and small-indel variant caller using deep neural networks. *Nature Biotechnology*, 36(10):983–987. 54, 131

[119] Porubsky, D., Vollger, M. R., Harvey, W. T., Rozanski, A. N., Ebert, P., Hickey, G., Hasenfeld, P., Sanders, A. D., Stober, C., Korbel, J. O., et al. (2022). Gaps and complex structurally variant loci in phased genome assemblies. *bioRxiv*. 130

[120] Ptashne, M. (2013). Epigenetics: Core misconcept. *Proceedings of the National Academy of Sciences*, 110(18):7101–7103. 15

[121] Rand, W. M. (1971). Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66(336):846–850. 95

[122] Rautiainen, M., Nurk, S., Walenz, B. P., Logsdon, G. A., Porubsky, D., Rhie, A., Eichler, E. E., Phillippy, A. M., and Koren, S. (2022). Verkko: telomere-to-telomere assembly of diploid chromosomes. *bioRxiv*. 43, 55, 129, 130

[123] Rivera, C. and Ren, B. (2013). Mapping human epigenomes. *Cell*, 155(1):39–55. 14

[124] Ruan, J. and Li, H. (2020). Fast and accurate long-read assembly with wtdbg2. *Nature Methods*, 17(2):155–158. 50

[125] Sagan, L. (1967). On the origin of mitosing cells. *Journal of theoretical biology*, 14(3):225–IN6. 48, 104

[126] Sánchez-Romero, M. A. and Casadesús, J. (2020). The bacterial epigenome. *Nature Reviews Microbiology*, 18(1):7–20. 14

[127] Sanger, F., Air, G. M., Barrell, B. G., Brown, N. L., Coulson, A. R., Fiddes, J. C., Hutchison, C. A., Slocombe, P. M., and Smith, M. (1977). Nucleotide sequence of bacteriophage ϕx174 dna. *Nature*, 265(5596):687–695. 150

[128] Scalzitti, N., Jeannin-Girardon, A., Collet, P., Poch, O., and Thompson, J. D. (2020). A benchmark study of ab initio gene prediction methods in diverse eukaryotic organisms. *BMC Genomics*, 21(1):293. 151

[129] Shearman, J. R., Sonthirod, C., Naktang, C., Pootakham, W., Yoocha, T., Sangsrakru, D., Jomchai, N., Tragoonrung, S., and Tangphatsornruang, S. (2016). The two chromosomes of the mitochondrial genome of a sugarcane cultivar: assembly and recombination analysis using long PacBio reads. *Scientific Reports*, 6(1):31533. 104

[130] Shi, Y., Liu, Y., Zhang, S., Zou, R., Tang, J., Mu, W., Peng, Y., and Dong, S. (2018). Assembly and comparative analysis of the complete mitochondrial genome sequence of sophora japonica 'jinhuaij2'. *PloS one*, 13(8). 104

[131] Shiro, I. (2012). A short history of the genome-wide association study: Where we were and where we are going. *Genomics Inform*, 10(4):220–225. 17

[132] Shumate, A. and Salzberg, S. L. (2021). Liftoff: accurate mapping of gene annotations. *Bioinformatics*, 37(12):1639–1643. 106, 130, 143

[133] Simão, F. A., Waterhouse, R. M., Ioannidis, P., Kriventseva, E. V., and Zdobnov, E. M. (2015). BUSCO: assessing genome assembly and annotation completeness with single-copy orthologs. *Bioinformatics*, 31(19):3210–3212. 39

[134] Simpson, J. T. and Durbin, R. (2010). Efficient construction of an assembly string graph using the FM-index. *Bioinformatics*, 26(12):i367–i373. 34

[135] Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J., and Birol, c. (2009). Abyss: A parallel assembler for short read sequence data. *Genome Research*, 19(6):1117–1123. 36

[136] Sloan, D. B., Wu, Z., and Sharbrough, J. (2018). Correction of persistent errors in arabidopsis reference mitochondrial genomes. *The Plant Cell*, 30(3):525–527. 124

[137] Stark, R., Grzelak, M., and Hadfield, J. (2019). Rna sequencing: the teenage years. *Nature Reviews Genetics*, 20(11):631–656. 151

[138] Sudlow, C., Gallacher, J., Allen, N., Beral, V., Burton, P., Danesh, J., Downey, P., Elliott, P., Green, J., Landray, M., et al. (2015). Uk biobank: An open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLOS Medicine*, 12(3):1–10. 48, 54

[139] Sullivan, A. R., Eldfjell, Y., Schiffthaler, B., Delhomme, N., Asp, T., Hebelstrup, K. H., Keech, O., Öberg, L., Møller, I. M., Arvestad, L., et al. (2020). The mitogenome of norway spruce and a reappraisal of mitochondrial recombination in plants. *Genome biology and evolution*, 12(1):3586–3598. 124

[140] Tosh, S. M. and Bordenave, N. (2020). Emerging science on benefits of whole grain oat and barley and their soluble dietary fibers for heart health, glycemic response, and gut microbiota. *Nutrition Reviews*, 78(Supplement_1):13–20. 25

[141] Trowsdale, J. (2005). Hla genomics in the third millennium. *Current Opinion in Immunology*, 17(5):498–504. Aging and the immune system / Immunogenetics / Transplantation. 49, 128

[142] Uffelmann, E., Huang, Q. Q., Munung, N. S., de Vries, J., Okada, Y., Martin, A. R., Martin, H. C., Lappalainen, T., and Posthuma, D. (2021). Genome-wide association studies. *Nature Reviews Methods Primers*, 1(1):59. 17

[143] Unseld, M., Marienfeld, J. R., Brandt, P., and Brennicke, A. (1997). The mitochondrial genome of Arabidopsis thaliana contains 57 genes in 366,924 nucleotides. *Nature Genetics*, 15(1):57–61. 48, 104

[144] Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A., et al. (2001). The sequence of the human genome. *Science*, 291(5507):1304–1351. 24

[145] von Luxburg, U. (2007). A tutorial on spectral clustering. 90

[146] Voss, S. R., Epperlein, H. H., and Tanaka, E. M. (2009). Ambystoma mexicanum, the axolotl: A versatile amphibian model for regeneration, development, and evolution studies. *Cold Spring Harbor Protocols*, 2009(8):pdb.emo128. 39

[147] Wang, R., Cai, X., Hu, S., Li, Y., Fan, Y., Tan, S., Liu, Q., and Zhou, W. (2020). Comparative analysis of the mitochondrial genomes of nicotiana tabacum: Hints toward the key factors closely related to the cytoplasmic male sterility mechanism. *Frontiers in Genetics*, 11. 112

[148] Wang, S., Song, Q., Li, S., Hu, Z., Dong, G., Song, C., Huang, H., and Liu, Y. (2018). Assembly of a complete mitogenome of chrysanthemum nankingense using oxford nanopore long reads and the diversity and evolution of asteraceae mitogenomes. *Genes*, 9(11):547. 104

[149] Wang, T., Antonacci-Fulton, L., Howe, K., Lawson, H. A., Lucas, J. K., Phillippy, A. M., Popejoy, A. B., Asri, M., Carson, C., Chaisson, M. J. P., et al. (2022). The human pangenome project: a global resource to map genomic diversity. *Nature*, 604(7906):437–446. 17

[150] Wang, Y., Zhao, Y., Bollas, A., Wang, Y., and Au, K. F. (2021). Nanopore sequencing technology, bioinformatics and applications. *Nature Biotechnology*, 39(11):1348–1365. 37

[151] Weber, J. L. and Myers, E. W. (1997). Human whole-genome shotgunsequencing. *Genome Research*, 7(5):401–409. 24

[152] Whiffin, N., Armean, I. M., Kleinman, A., Marshall, J. L., Minikel, E. V., Goodrich, J. K., Quaife, N. M., Cole, J. B., Wang, Q., Karczewski, K. J., et al. (2020). The effect of lrrk2 loss-of-function variants in humans. *Nature Medicine*, 26(6):869–877. 48, 54

[153] Woloszynska, M. (2010). Heteroplasmy and stoichiometric complexity of plant mitochondrial genomes?though this be madness, yet there's method in't. *Journal of experimental botany*, 61(3):657–671. 104, 119, 124

[154] Woloszynska, M. and Trojanowski, D. (2009). Counting mtdna molecules in phaseolus vulgaris: sublimons are constantly produced by recombination via short repeats and undergo rigorous selection during substoichiometric shifting. *Plant Molecular Biology*, 70(5):511–521. 119, 124

[155] Wooster, R., Bignell, G., Lancaster, J., Swift, S., Seal, S., Mangion, J., Collins, N., Gregory, S., Gumbs, C., Micklem, G., et al. (1995). Identification of the breast cancer susceptibility gene brca2. *Nature*, 378(6559):789–792. 15

[156] Wu, Z., Hu, K., Yan, M., Song, L., Wen, J., Ma, C., Shen, J., Fu, T., Yi, B., and Tu, J. (2019). Mitochondrial genome and transcriptome analysis of five alloplasmic male-sterile lines in brassica juncea. *BMC genomics*, 20(1):348. 104

[157] Yamada, T. and Kinoshita, H. (2002). Finding all the negative cycles in a directed graph. *Discrete Applied Mathematics*, 118(3):279–291. 70

[158] Yan, F., Powell, D. R., Curtis, D. J., and Wong, N. C. (2020). From reads to insight: a hitchhiker's guide to atac-seq data analysis. *Genome Biology*, 21(1):22. 151

[159] Yang, C., Chu, J., Warren, R. L., and Birol, I. (2017). NanoSim: nanopore sequence read simulator based on statistical characterization. *GigaScience*, 6(4). gix010. 95

[160] Yoshimura, J., Ichikawa, K., Shoura, M. J., Artiles, K. L., Gabdank, I., Wahba, L., Smith, C. L., Edgley, M. L., Rougvie, A. E., Fire, A. Z., et al. (2019). Recompleting the caenorhabditis elegans genome. *Genome Research*, 29(6):1009–1022. 42

[161] Zerbino, D. R. and Birney, E. (2008). Velvet: Algorithms for de novo short read assembly using de bruijn graphs. *Genome Research*, 18(5):821–829. 36

[162] Zou, Y., Zhu, W., Sloan, D. B., and Wu, Z. (2022). Long-read sequencing characterizes mitochondrial and plastid genome variants in arabidopsis msh1 mutants. *The Plant Journal*, 112(3):738–755. 104, 119, 124

Assembly of nucleotide sequences of the multipartite structures in plant mitochondria genomes and the diploid human genomes
(植物ミトコンドリアゲノムおよび 2 倍体ヒトゲノムの塩基配列のアセンブリ)

Bansho Masutani
舛谷万象

December 7, 2022