Freie Universität Berlin

De novo and haplotype assembly of polyploid genomes

Mohammadhossein Moeinzadeh

Dissertation zur Erlangung des Grades
eines Doktors der Naturwissenschaften (Dr. rer. nat.)
am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

Berlin, 2018

Erstgutachter: Prof. Dr. Martin Vingron
Zweitgutachter: Prof. Dr. Bernd Weisshaar

Datum der Disputation: 10.12.2018

To Sarah, Sheyda, and Kian

Acknowledgements

First and foremost, I would like to thank my supervisor Martin Vingron for his scientific support and guidance during my PhD. The door to his office was always open whenever I ran into a trouble spot or had a question about my research. I am grateful to him for giving me the opportunity to pursue my PhD in his group with exceptional scientists and a friendly environment.

My sincere thanks also go to Jun Yang, who contributed to the design of the analysis described in this thesis. I appreciated his help and time during the course of my study as a PhD candidate.

Besides my advisor, I would like to thank the rest of my thesis advisory committee: Stefan Haas and Knut Reinert for their insightful advice for my thesis.

I thank Kirsten Kelleher for her kind help and support as PhD coordinator of IMPRS program, for the help with settling in Berlin, and especially for the great help of proofreading this thesis.

I thank Peter Arndt, Sebastiaan Meijsing, Robert Schöpflin, David Heller, and Evgeny Muzychenko for the help and insightful discussions and also my former office mates Jun Yang, Xinyi Yang, Mohammad Sadeh, and Emmeke Aarts and all the current and former members of the Computational Molecular Biology group at MPIMG for the friendly and motivating environment.

I would like to thank my parents for supporting me spiritually throughout my study and my life in general. Finally, I must express my very profound gratitude to my wife for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without her. Thank you.

Contents

Abstract

Sequencing and genome assembly constitute the basis for further research into the biology of an organism; however, the main players are chromosomes, not the consensus references. The assembly programs flatten two or more homologous chromosomes, depending on the ploidy of the organism, into one reference sequence. Although the major characteristics and functionalities of homologous chromosomes are the same, the minor differences may play an important role. The sequence of one chromosome is called a *haplotype*. A haplotype is the main component of inheritance; hence, obtaining the haplotype sequences instead of the consensus, provides a panoramic view for investigations. Human is diploid and has two copies of each autosomal chromosome, each inherited from one parent. Other organisms have several copies of the autosomal chromosomes. Higher ploidy of an organism leads to more information being flattened in the reference sequence, and results in less similarity between the reference sequence and any of the chromosomes.

In this thesis, we focus on haplotyping problems and various approaches to call the haplotypes. In the first chapter, *Genotype and haplotype*, the approaches for inferring haplotypes and the challenges are discussed. In *Chapter 3*, we focus on single individual haplotyping based on the sequence reads, which is currently the major and most direct approach to haplotyping. Different technologies, protocols, models, and methods are also reviewed. In *Chapter 4*, we propose a novel method, called ***Ranbow***, to address the polyploid haplotype assembly problem. The performance of ***Ranbow*** compared to the other state-of-the-art methods is investigated on both real and simulated datasets. For the evaluation, we used data from tetraploid *Capsella bursa-pastoris* (Shepherd's Purse), and hexaploid *Ipomoea batatas* (sweet potato) genomes (*Chapter 5*). In *Chapter 6*, one application of reconstructed haplotypes is discussed. We assembled the sweet potato genome and used haplotype information to propose a novel scaffolding method called haplo-scaffolder. Finally, a summary of the thesis is provided and possible future directions are discussed in the summary chapter.

Chapter 1

Introduction

The idea of genome assembly with sequence reads goes back to 1979 when Rodger Staden mentioned: "With modern fast sequencing techniques and suitable computer programs it is now possible to sequence whole genomes without the need of restriction maps"[1]. Although at the time of this quote, the "modern sequencing techniques" were gel reading, and the "computer programs" were stored on tapes, the underlying principle for genome assembly has stayed the same: genome assembly based on reads' overlaps. From then on, with first generation sequencing technology, new tools for the genome assembly were developed. Virus genomes of a few kilobases were assembled by Sanger *et al.* in 1978[2], and then, 1Mb bacterium (1995)[3], 120Mb fruit fly (2000)[4], and 3Gb human (2001)[5] genomes were sequenced and assembled.

With the advancement of sequencing technologies and assembly tools several consortia and groups have been addressing human and non-human genomes; however, the assembled references do not provide a complete picture of the underlying biology: the main entities are chromosomes, not the consensus references sequences; a reference is an estimate for the real entity behind it. The sequence of one chromosome is called the *haplotype* (*Section 2.2*). Haplotypes are the main components of inheritance, and most of the chromosomal regulatory interactions are derived within haplotypes. Hence, obtaining the haplotype sequences, instead of the consensus, provide us with a panoramic view for the investigations. *Section 2.3* explains the importance of haplotype assembly. Human is diploid and has two copies of each autosomal chromosomes, each inherited from one parent. Various organism may have several copies of autosomal chromosomes. For instance, treefrog[6] is triploid (three copies, or 3x), zebrafish[7], axolotl[8], and potato[9] are tetraploid (4x), wheat[10] and sweet potato[11] are hexaploid (6x), and strawberry[12] is octaploid (8x). The higher ploidy of an organism causes more information to be flattened in the reference sequence, and results in less similarity between the reference sequence and any of the chromosomes. Several approaches have addressed the haplotyping problem based on different data, such as haplotyping via family pedigree, population data, or from sequence reads (*Section 2.4*).

In single individual haplotype reconstruction based on reads, those reads that span at least two variants are the informative ones. This restriction on the reads' property makes the haplotyping problem challenging dependent on the sparsity of the variants. It took six years, after the first release of human genome, to publish the first human haplotype, called *HuRef*. Part of the reason was the sparsity of the variants (it is in the range of one in 1kb). *Section 2.5* refers to challenges for diploid and polyploid haplotyping. The HuRef assembly was done with Sanger paired sequencing which provides relatively long reads with high accuracy; which potentially can cover more than one polymorphic site (*Section 3.3*). The focus has been shifted, owing to the high cost of Sanger sequencing, by emerging high-throughput next generation sequencing (NGS). But the NGS read length was, and still is, too short to cover two or more variants needed for haplotyping. Since then, several different types of data integration (*Section 3.4*), library preparation, and recruiting single cell technologies (*Section 3.5)* have been proposed. Fosmid, HiC, Strand-seq, and 10X Genomics library preparations and chromosomal isolation before sequencing are among these approaches. The single molecule real-time (SMRT) technologies bring the sequencing technologies to the third generation by producing long but error prone reads. Although, SMRT reads span several variants, the high error rate makes the variant calling and pinpointing the variants in reads challenging. Integration of SMRT and NGS technologies is one solution for dealing with such a problem (*Section 3.6*).

All these attempts have addressed the human genome, or can be applied for any diploid genomes; however, the polyploid genome and haplotype assembly are more challenging. The high heterozygosity that comes from the presence of three or more copies of the monoploid genome will always hinder the genome assembly process; the reason being that genome assembly focuses on the vast majority of bases that are invariant across homologous chromosomes. Since these invariant regions are intermittent along the whole chromosome, the result is fragmentation in polyploid assembly. Any error in the reference is propagated to variant calling and read mapping, which are the inputs for haplotyping. In addition, the main principle for sequence assembly, read assembly by proper overlaps, is violated in polyploid haplotype assembly. However, the high heterozygosity of these genomes may come to help haplotyping with short reads; the higher heterozygosity, the higher the chance of covering two variants with one short read. A few methods have been reported for polyploid haplotype assembly, but due to lack of a gold standard data set, they have not been applied on real genomic data, and are not scalable to deal with billions of short reads generated for *de novo* assembly (*Section 3.7*).

We proposed a novel method to address the polyploid haplotype assembly problem. It accepts the reference sequence, mapped reads, and called variants as input and produces a set of mapped assembled haplotypes. This method is called ***Ranbow*** and is explained and evaluated in *Chapter 4* and *Chapter 5*, respectively. ***Ranbow*** initially was developed for the haplotype assembly of sweet potato and for the improvement of its assembled genome; however, it can be applied to other polyploid genomes as well. We assessed ***Ranbow*** on simulated and real datasets. The simulated datasets are inspired by the hexaploid sweet potato and the tetraploid *Capsella bursa-pastoris* (shortly CBU) genomes. We tested the performance of ***Ranbow*** by changing different parameters such as sequence read insert sizes. For real dataset, Roche 454 sequence reads were produced and served as the ground truth dataset to evaluate the assembled haplotypes from Illumina short reads. The characteristics of these datasets and the results are explained in *Chapter 5*.

In *Chapter 6*, we explain *de novo* assembly of the sweet potato genome which was done in collaboration with Jun Yang, an independent researcher in our lab also affiliated to Chinese Academy of Sciences, and the sequencing core facility of Max Planck Institute for Molecular Genetics. DNA material was transferred from China and sequenced in house. The initial *de novo* assembly was done in the sequencing core facility by Heiner Kuhl and provided to us for further analysis and improvement. We proposed a new scaffolding tool, called haplotype-aided scaffolder. The proposed method relies on the assembled haplotypes and the connections between scaffolds which are inferred through the reads mapped to these haplotypes. This method improved the N50 scaffold size of sweet potato by $\sim$40%. We suggest haplotype-aided scaffolding as a step to improve *de novo* assembly of complex genomes. This collaboration resulted in a report published in Nature Plants[11].

Chapter 2

Genotype and Haplotype

2.1 Introduction

In this chapter, firstly, we define haplotypes and relevant terminology (*Section 2.2)*. Then, we explain how the availability of haplotypes for both diploid and polyploid genomes is of a great importance (*Section 2.3)*. There are a number of different approaches for obtaining haplotypes depending on the type of input data, such as population data, relative genotypes, and sequence reads of a single individual. These approaches are listed and discussed in *Section 2.4*. In this thesis, we focus on single individual haplotyping from sequence reads; hence, a number of potential challenges for single individual haplotyping are mentioned in *Section 2.5*. In the last two sections of this chapter, the availability of ground truth data is discussed (see *Section 2.6)* and then the metrics for evaluation of computed assembled haplotypes are described (see *Section 2.7)*.

2.2 Genotype and Haplotype

The human genome is diploid, it has two sets of autosomal chromosomes. Each copy is inherited from one parent. The two copies of an autosomal chromosome are similar, yet the information they carry differs slightly. These differences are called sequence variants or alleles and are located in the polymorphic sites (or loci) in the genome. They can be **Single Nucleotide Polymorphism** (SNP, when the variant is observed in more than 1% within a population), **Single Nucleotide Variant** (SNV, when it appears in $< 1\%$ within a population), indels (insertion or deletion when the region in question is $< 50bp$ in size) and structural variants (deletion, duplication, insertion, and translocation). The set of variants at all polymorphic sites is called the **genotype**. However, genotype does not provide the complete genetic map for an organism, as the homologous chromosomes are flattened into one reference sequence and its sequence variants, without information about the origin of these variants. The set of variants that are located on a single chromosome is called **Haplotype**[1] (*Figure 2.1)*. In contrast to diploid organisms, polyploid organisms have more than two copies of each autosomal chromosome. These genomes have undergone the duplication of entire genomes[13] resulting in more than two copies of each chromosome. Although polyploidy in animals is less frequent than in plants, there are several insects, amphibians, reptiles and fish whose genomes are reported as polyploid[13].

In order to explain the meaning of haplotypes, let's define the following terms more precisely (*Figure 2.1* illustrates these terms in an example). Note that some of the following terms are defined for a population of individuals while the other terms are explained for a single individual:

- **Polymorphism:** The coexisting of different alleles in the individuals of the same population.
- **Polymorphic site:** A genomic position whose corresponding alleles may vary among the chromosomes in the population. A polymorphic site may point to two (biallelic polymorphic site) or more (multiallelic) variants which are detected in the population.
- **SNP:** Single nucleotide polymorphism, or SNP, is a polymorphic site containing DNA sequence variations in the size of one single base pair which its variant is observed in more than 1% of population.
- **Indel:** Insertion and deletion smaller than 50 base pair.
- **Multinucleotide polymorphism:** The variants which are larger than one basepair and are equal in length *e.g.* 'ATC' and 'CGA'.
- **SmP:** In this thesis, we introduce Small Polymorphism, shortly SmP, which could be a single nucleotide variant, a multinucleotide variant, or an indel. SmP covers all type of small variants which are dealt with in this thesis.
- **Interpolymorphic region:** An interpolymorphic region is a region between two consecutive polymorphic sites. The sequences of nucleotides in these regions are identical among homologous chromosomes.
- **Homologous chromosomes:** A group of chromosomes with a similar length and centromere position, which possess similar genes at corresponding loci. For example, human is diploid having 22 pairs of homologous autosomes. A P-ploid

[1]Frequently used terms are appeared in blue and explained in glossary.

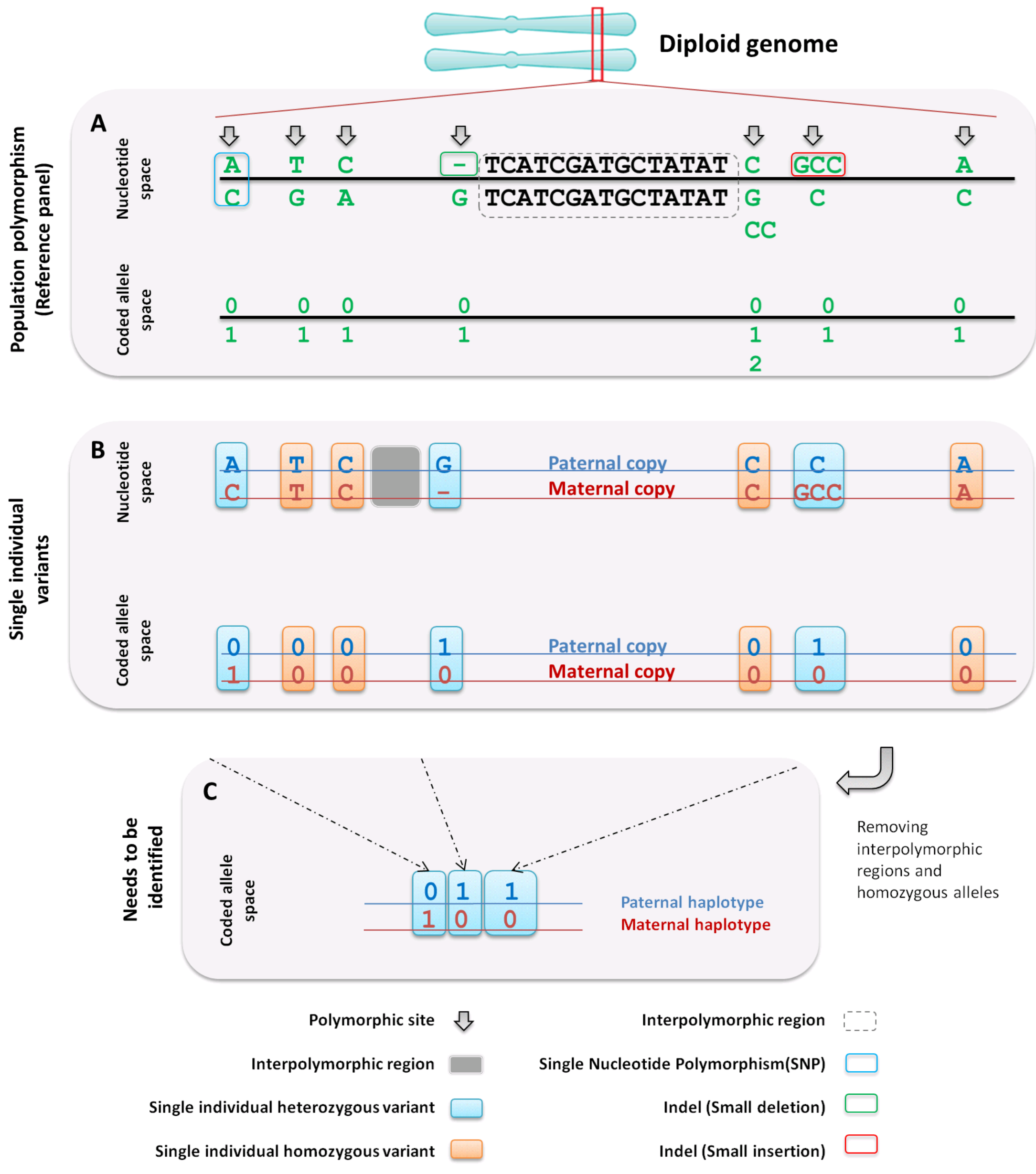

Diploid genome
Population polymorphism (Reference panel)
A
Nucleotide space
A T C – TCATCGATGCTATAT C GCC A
C G A G TCATCGATGCTATAT G C C
CC
Coded allele space
0 0 0 0 0 0 0
1 1 1 1 1 1 1
2
Single individual variants
B
Paternal copy
Maternal copy
Needs to be identified
C
Paternal haplotype
Maternal haplotype
Removing interpolymorphic regions and homozygous alleles
Polymorphic site
Interpolymorphic region
Single individual heterozygous variant
Single individual homozygous variant
Interpolymorphic region
Single Nucleotide Polymorphism(SNP)
Indel (Small deletion)
Indel (Small insertion)

FIGURE 2.1: **Schematic view of genotype and haplotype.** Panel A or reference panel illustrates a reference sequence (black line) and its polymorphic sites identified in population. These polymorphic sites are then depicted as genotype in the coded allele illustration. Panel B shows polymorphic sites in a single individual consisting of hetero- and homozygous variants. These variants are depicted in both nucleotide- and coded allele spaces. In order to infer the coded allele from a variant, one has to check how the variant is coded in the reference panel and the genotypes. Panel C shows the haplotypes in coded allele space. Since the interpolymorphic, *i.e.*, the regions between consecutive polymorphic sites, and homozygous regions are identical in both chromosomes, their sequences in parental haplotypes can be simply inferred from the reference sequence. The objective of haplotype reconstruction is to infer the connection of heterozygous variants. Panel C shows what haplotype reconstruction methods are aiming for.

genome consists of a set of chromosomes which contains P homologous chromosomes. For instance, sweet potato has a hexaploid genome (P=6) with the base chromosome set of 15 meaning that there are 15 sets of homologous chromosomes each of which contains six chromosomes ($15 \times 6 = 90$ chromosomes).

- **Sequence variant:** A sequence appears in one chromosome of one individual at a polymorphic site. It is worth noting that a sequence variant could be a SNP or indel. In this thesis, we use **variant** and sequence variant interchangeably.
- **Genotype:** A set of variants at a polymorphic site is called a genotype. For example in *Figure 2.1-A*, the first genotype is $\{A, C\}$ and the second one is $\{T, G\}$. The genotypes are illustrated as unordered sets, meaning that it is not clear which allele belongs to which of the parental chromosomes.
- **Homozygous variant:** A homozygous variant is a variant which is identical in all homologous chromosomes. Note that when we talk about these variants, we are referring to one single individual and not the population. There are other variants in the population at the corresponding polymorphic site but the individual carries the same variant in all of its homologous chromosomes.
- **Heterozygous variant:** The sequence variants which are not identical among the homologous chromosomes of a single individual.

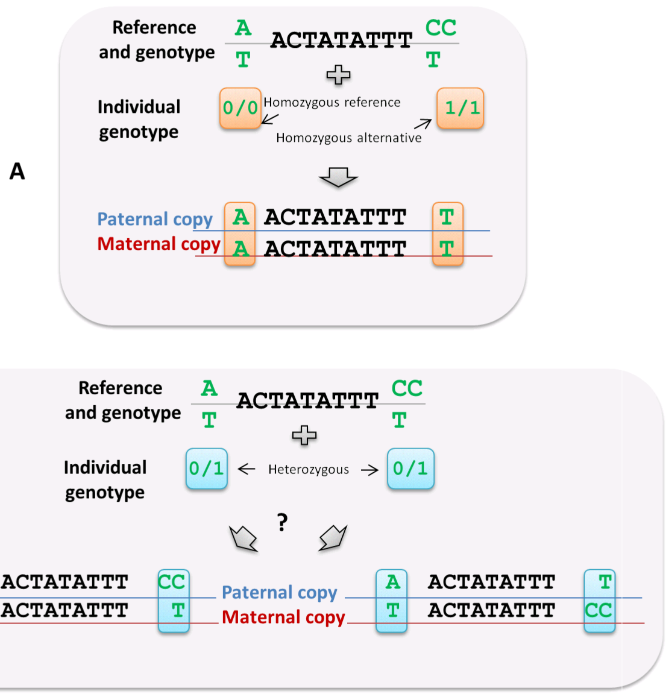


FIGURE 2.2: **Homozygous and heterozygous variants.** A) two consecutive homozygous variants, one reference and one alternative. Obtaining haplotypes is trivial and there is no ambiguity in inferring them. B) Ambiguity of obtaining the haplotypes when the polymorphic sites are heterozygous. There are two possibilities for the haplotypes considering two consecutive heterozygous polymorphic sites.

The sequences of nucleotides in the regions between polymorphic sites are identical in autosomal chromosomes. The same applies to homozygous variants. Since the sequences of these variants are identical, the variants in each of the chromosomes are known, and they do not provide information for phasing the other variants. Therefore, the focus of haplotyping is on the heterozygous variants. *Figure 2.2-A* indicates the sequence of haplotypes are known when the polymorphic sites are homozygous. *Figure 2.2-B* illustrates the ambiguity of connection between alleles caused by two consecutive heterozygous variants. In order to solve this ambiguity, extra information such as sequence reads sampled from one chromosome is demanded. Therefore, the ordered collection of heterozygous variants of each chromosome together with the reference sequence provide a complete picture of a haplotype. So, the haplotyping problem for a $P-$ploid genome, $P = 2$ for diploid and $P > 2$ for polyploid genomes, is formulated as finding the ordered sequence variants in one chromosome (*Figure 2.1-C*). These variants are coded into numbers for simplicity; 0 is used for the reference allele, while $1, 2, 3, \ldots, P-1$ are used for the alternative ones. Hence, in a more abstract form, one haplotype is a sequence of numbers with the length of the number of heterozygous polymorphic sites. We call this illustration, haplotypes in coded allele space (see *Figure 2.1*).

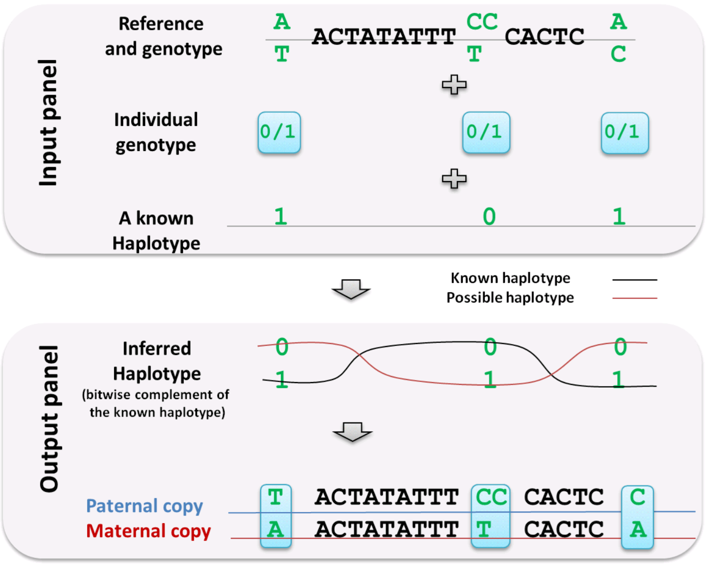


FIGURE 2.3: **Obtaining haplotypes of a diploid genome once one of the haplotypes is known.** When one haplotype is known, the other haplotype can be simply computed as its bitwise complement.

For a diploid genome, the alleles in one haplotype are either the reference allele or the alternative one; therefore, the two haplotypes are the bitwise complement of each other. Recall that the homozygous alleles are unambiguous and are removed. In other words, knowing one haplotype sequence gives us the other haplotype sequence as well. This, in a sense, makes the computational problem definition of diploid and polyploid haplotyping different. For diploid genomes, the aim is obtaining just one haplotype sequence. Once one haplotype is inferred or assembled, the other haplotype can be computed by converting zeros to ones and ones to zeros (*Figure 2.3*). However, for polyploid genomes, this principle cannot be employed. *Figure 2.4* shows four possible groups of haplotypes which can be inferred if one of the haplotypes is known and given as input. This indicates that all of the P haplotypes in P-ploid genome need to be inferred or assembled, which distinguishes the computational problem for reconstructing diploid and polyploid haplotypes. In diploid genomes, we need to assemble one haplotype (not two), but in polyploid one all P haplotypes need to be assembled.

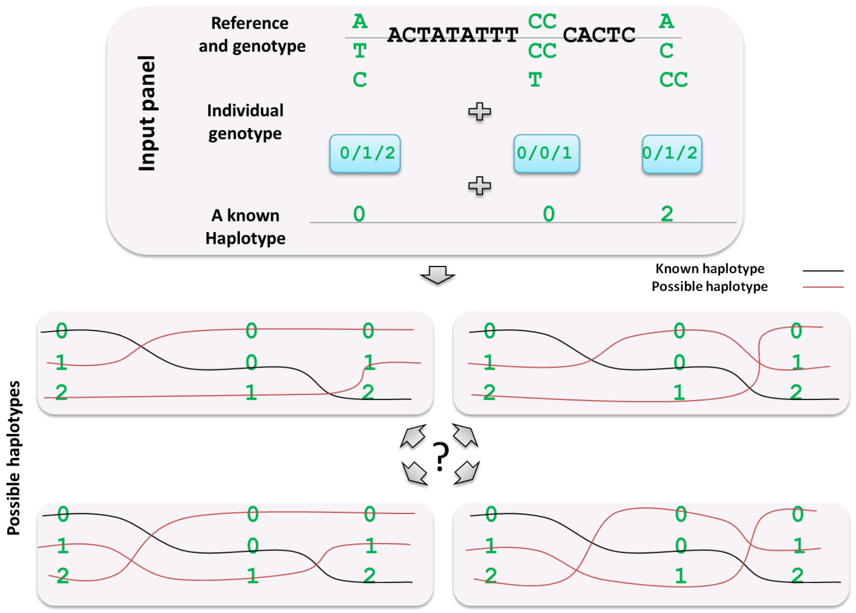


FIGURE 2.4: **Obtaining haplotypes of a polyploid genome once one of the haplotypes is known.** This figure shows a triploid genome and its genotype for three heterozygous positions. Given the genotype and one of the haplotypes, there are four possible ways for obtaining the other two haplotypes. This figure shows these four possible haplotype groups.

We define the following technical terminology (see *Figure 2.1* for an illustration):

- **Haplotypes in nucleotide space:** A haplotype shown as a sequence of nucleotides.
- **Haplotypes in coded allele space:** As it is mentioned, haplotypes can be defined as the sequence of coded alleles on heterozygous polymorphic sites. This representation in this thesis is called haplotypes in coded allele space. Given a haplotype in coded allele space, one can decode the numbers into the corresponding alleles from the genotype and then insert the homozygous alleles and inter-polymorphic regions with the guide of reference sequence in order to obtain the haplotype in nucleotide space.

- **Fragment:** A read in coded allele space is called a fragment.
- **Genotype in coded allele space:** A genotype is an unordered set so the illustration should include this property; hence, the genotypes are shown as a set of numbers separated with slashes. For example, '0/0/1' indicates that the genotype consists of two reference alleles and one alternative one. Homozygous and heterozygous genotypes are illustrated in *Figure 2.2-A* and *Figure 2.2-B* respectively.
- **Missing allele:** This terminology, which is illustrated as '$-$', is used when an allele is unknown in coded allele space. For example '$10210 - 21$' is a haplotype where its sixth allele is unknown. Since this dash is used in coded allele space, there is no danger of confusing it with a deletion.

Note that, when the base pairs are illustrated and used, it implicitly indicates that the haplotypes are in nucleotide space while using a sequence of numbers ($0 \leq$numbers$<$P) indicates that the haplotype is in coded allele space.

2.3 The importance of haplotypes

Haplotype information plays an important role in diverse contexts, such as disease association studies, population genetics, and also in technical and computational problems like *de novo* assembly of a genome. Here, we list some of these applications as follows.

2.3.1 Disease association studies

Compound heterozygosity: Inferring haplotypes is essential for addressing several problems of molecular biology. Finding the **compound heterozygosity** is one of the applications of the phased genome, i. e. a genome with known haplotypes (*Figure 2.5-A*). Compound heterozygosity occurs when two mutations hit one gene in two different positions. The two mutations may exist either in the same copy (**cis-compound heterozygosity**) or in both paternal and maternal alleles, each at a different position (**trans-compound heterozygosity**). In the trans case, both alleles may behave abnormally and cause a disease. For instance, in the case of Miller syndrome, Roach *et al.*[14] studied a patient with the syndrome, his sibling and their parents. They sequenced the four individuals' genes and found that there is one gene which harbours two mutations in two sites; one in the maternal and another one in the paternal copy. This information is not accessible through genotyping. The genotype just shows that the two mutations are there, but it does not indicate if the two hits are in one haplotype (cis) or in different ones (trans).

Variants in regulatory elements: The same principle, mentioned above, is valid for distant enhancers and the corresponding genes. The chromosomes are packed in a 3D structure, which brings the regulatory elements from far physical distance in genomic DNA to close proximity in 3D space. It has been shown that a majority of the interactions between genes and regulatory elements are intra-chromosomal[15]. For example, having two variants in a gene and its enhancer (shown in *Figure 2.5-B*) leads, again, to the cis and trans scenarios. In the former, there is a copy with no variant and the expression of one copy may compensate the effect of mutation in the other gene maintaining the phenotype[16].

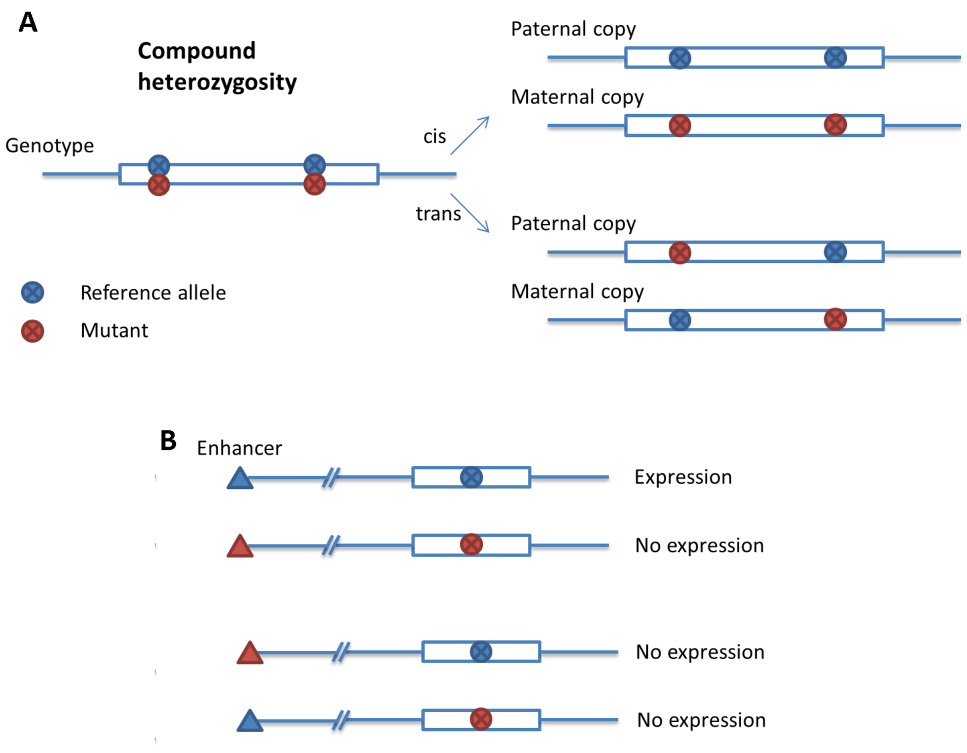


FIGURE 2.5: **A) Compound heterozygosity in a gene. B) Compound heterozygosity in one gene and its enhancer**

High frequency and low effect variants: Compound heterozygosity and the studies mentioned above are about highly penetrant variants, *i.e.* variants which are highly associated with the traits and phenotypes. However, higher frequency of variants which have a low effect could cause a disease as well. In the case of systemic lupus erythematosus (SLE), Graham *et al.*[17] reported variants in a protein coding gene (tumour necrosis factor α-protein 3 - TNFAIP3) as well as in two haplotype blocks, one 200kb upstream and one 200kb downstream of the gene. Therefore, not just one variant but the combination of all variants in this $\sim$400kb region may play a role in this disease.

Pharmacogenetics: In addition to the investigations on the causal variants in diseases, haplotype information plays a role in clinical genetics and pharmacogenetics. A different combination of variants in one haplotype could result in different protein sequences and may alter the metabolism the proteins are involved in; hence, knowing the phase helps to predict the drug response of the patient[18].

2.3.2 Population genetics and evolution of organisms

Haplotypes are the units of inheritance and provide higher precision than genotypes for population genetics studies. Population history, migration, and bottlenecks can be traced back more confidently with haplotype resolution[19]. It is possible to trace back the evolutionary history of an organism considering the haplotype blocks. In Yang *et al.*[11], we reported the estimation of polyploidization time via assembled haplotype blocks in the sweet potato genome. Sweet potato is hexaploid, with six copies of each chromosome. A haplotype block in this context is a region which is phased into its six haplotypes. Each haplotype block reveals an evolutionary model for the polyploidization. Inferring a model for each block revealed the most probable historical evolutionary model and its timing for this genome.

2.3.3 Sequence assembly and variant calling for polyploid genomes

Variant calling improvement: Haplotype information can be used in a post processing step for variant callers, *i.e.* the tools that identify the existence of variants at polymorphic sites from sequence reads. Given haplotype data, variant callers not only consider the allele frequency in polymorphic sites but also integrate the information of the neighbouring alleles. For instance in a hexaploid organism like sweet potato, three consecutive SmPs with two, two, and three alleles, which are called by a variant caller, could give rise to $2 \times 2 \times 3 = 12$ haplotype combinations. Just six out of these 12 combinations are correct. The assembled haplotype support for each combination is taken into account, and the alleles are evaluated considering haplotypes[11]. This application will be explained more in *Chapter 6*.

***De novo* assembly improvement:** Haplotypes are of great importance for *de novo* assembly of highly heterozygous genomes. The high heterozygosity, which partly comes from the presence of three or more copies of each chromosome, always hinders the genome assembly process even with state-of-the-art assembling tools, because genome assembly focuses on the vast majority of bases that are invariant across homologous chromosomes (interpolymorphic regions). Since these regions are smaller along the whole chromosomes of polyploid genome due to high heterozygosity, the assembly is more fragmented. Phasing the haplotypes splits the variant regions into their original sequences which the assembler can rely on to deal with the fragmentation[11]. More details can be found in *Chapter 6*.

2.4 Different approaches for haplotyping

Several techniques, protocols, and computational approaches have been reported for inferring the haplotypes, most of them addressing the human genome. These methodologies are, therefore, more compatible with the characteristics of the human genome, diploid with low-density of alleles. These approaches are divided into the following main categories based on the data and the experimental settings.

2.4.1 Haplotype phasing via family information

This approach, which is also called **genetic haplotyping**, or **haplotyping with pedigree information**, combines genotype and pedigree data to infer haplotypes. In this approach, the genotypes are mainly obtained from SNP arrays with predefined primers, therefore, the sequences of the SNPs need to be known at first to design the primers. This strategy works when the SNP is already known, but it misses the SNVs. The pedigree information shows the relation between members of a family. Phasing a trio, a child and the parents, shows a successful result on the sites which contain both heterozygous and homozygous alleles within family members. For instance, when a/A, A/A, and a/A are the genotypes of the father, mother, and child respectively, it is clear that a (A) is transmitted from father (mother). Trio phasing fails when all sites are equally heterozygous, all genotypes are equal. In this case, the genotypes of other relatives may be useful. Working with genotype data is limited to the known variants, and it is not possible to phase SNVs sites.

2.4.2 Haplotype phasing via population data

These methods rely on the shared haplotype blocks within a population which are obtained from the genotype information and the linkage disequilibrium (LD) measure. The LD measure is calculated based on the association of variants in different position in a given population. To obtain this, the frequency of combinations of variants are measured and then compared to a scenario in which variants are inherited together at random. The shared blocks can be exploited to phase all individuals in the population including the target one. Based on a model, such as the assumption of the minimum number of haplotype sequences in a population (parsimony model) [20], the haplotypes are phased. This approach is effective for imputing missing alleles, *i.e.*, estimating the missing alleles with the help of flanking variants and their frequency in population data, and phasing the common variants, and can result in haplotype segments up to 300kb long in size[21]; however, this approach is limited when dealing with rare variants and crossing recombination hotspots [21].

2.4.3 Single Individual Haplotyping (SIH)

This group of methods aims to assemble the haplotypes from sequence reads of a single individual. Each read is sampled from one chromosome and can serve as a short haplotype. These approaches rely on the sequence overlaps on polymorphic sites and glue the reads based on the allele similarity to obtain a longer haplotype (see *Figure 2.6*). The methods for SIH have developed along with sequencing technologies. Firstly, the mate-pair Sanger reads were used for this purpose. These reads are long enough to connect consecutive alleles in most of the regions in the human genome, but the sequencing technology is costly. The next generation sequencing (NGS) technologies provide high-throughput short read data. Most of the current NGS reads are not usable for low-heterozygous genomes like the human one. However, in the more heterozygous genomes the polymorphic sites are closer, and, therefore, haplotyping with the short NGS data may also be possible. In Yang *et al.*, we proposed an approach to deal with complex and highly heterozygous genomes with NGS data, that involves Illumina sequencing (100bp and 150bp reads) data on the heterozygous sweet potato genome[11].

To address human haplotyping, several approaches integrate NGS data with other sequencing technologies or protocols, such as HiC, Strand-seq, RNA-Seq and chromosome isolation. Third-generation sequencing outputs long (several kilobase pairs) sequences sampled from one chromosome. These type of reads not only connect several polymorphic sites, but could span problematic genomic regions like repeats. In addition, 10X Genomics produces linked-read data, *i.e.* highly accurate short reads that are sampled from one high molecular weight (HMW) fragment. These approaches will be explained in more detail in *Chapter 3*.

2.5 Challenges

Unlike genome assembly that focuses on the majority of invariant bases in the reads, haplotype reconstruction methods need variants. The genome properties, such as variant

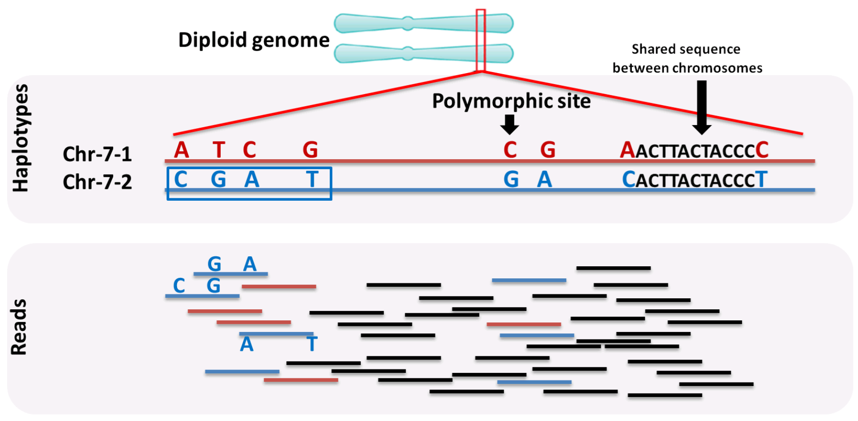

Diploid genome
Shared sequence between chromosomes
Polymorphic site
Haplotypes
Chr-7-1
Chr-7-2
A T C G C G AACTTACTACCCC
C G A T G A CACTTACTACCCT
Reads
G A
C G
A T

FIGURE 2.6: **Haplotype reconstruction for a diploid genome.** As an example, a part of human chromosome seven is depicted in this figure. The variants in polymorphic sites of each homologous chromosome (red and blue) are illustrated in the same color as the chromosomes. The lower part of the figure shows the sequence reads, and the variants they carry. These reads are partly sampled from the blue chromosome and partly from the red one. The reads can be grouped according to the information they carry. A read is useful for haplotype assembly if it contains at least two sequence variants. Reads with no or one sequence variant are not useful and excluded for haplotyping (black reads).

distribution, as well as sequencing features, like read length and accuracy, play a major role in haplotyping. Here we list a number of genome properties and technical issues which are important for diploid and polyploid haplotype assembly.

2.5.1 Genome ploidy

Methods for haplotyping are, from a technical point of view, divided into two groups of diploid and polyploid haplotypers. The reason, as explained in *Section 2.3*, is that the haplotypes in diploid genomes are the bitwise complement of each other. The availability of both alleles in heterozygous sites, and knowing the sequence of one haplotype directly result in the sequence of the other one. Hence, the goal of diploid haplotype assembly can be redefined as the assembly of one of the haplotypes. Likewise, the situation with two overlapping reads with an identical overlap indicates that the two reads are sampled from one haplotype. This can be considered as a foundation in diploid haplotyping. However, polyploid genomes do not have this property. Knowing the sequence of one haplotype provides no information about the others. Accordingly, it can change the problem definition in the assembly of reads as well. The matching overlap between two aligned reads is not sufficient to infer whether they belong to one haplotype. *Figure 2.7* illustrates this situation. More information is needed to decide whether two reads are from one haplotype and can be merged. We call this property **Ambiguity of Merging Problem** (AoM problem) (*Figure 2.7*).

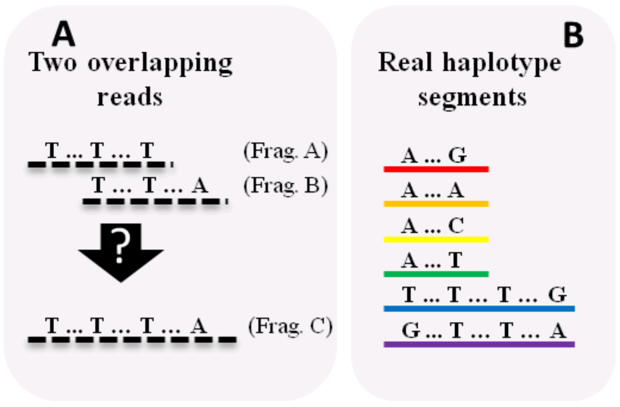


FIGURE 2.7: **AoM problem** The fragments A and B have an overlap with two matching SmPs ($T ... T$). Frag.C is made from merging Frag.A and Frag.B. The real haplotype sequences from which Frag.A and Frag.B are sampled, are illustrated in (B), showing that Frag.C does not belong to any of the real haplotypes. This ambiguity makes it difficult to merge two fragments with high confidence, only by considering their sequences.

2.5.2 Low variant density

The higher heterozygosity of polyploid genomes can be helpful in polyploid haplotyping. The smaller the distance between polymorphic sites, the more variants present in one sequence read, and the more useful reads there are for haplotyping. *Figure 2.8* shows the comparison between the proportion of reads that are usable for haplotyping in human and the same for sweet potato. In general, the average interval distance of polymorphic sites in human and sweet potato is ∼1kb and ∼60bp, respectively.

2.5.3 Structural variants

Structural variants affect the haplotyping since they may change the number of sequence copies. One unrecognized duplication inflates the number of reads which are mapped to the reference in the duplicated region. If the duplicated sequences diverge enough, they may accumulate a number of variants. These variants, together with the original variants, are mapped to the corresponding region in reference. This may affect the haplotyping since the haplotypers assume the number of copies is the same as the ploidy, but in reality, there is one more copy available (*Figure 2.9*). One potential approach to address this problem is finding these regions and applying the haplotype assembler with the proper ploidy. However, finding structural variants is another problem and needs its own consideration. The same reasoning is also applied to paralogous genes. Without knowing the structural variants, more careful consideration is necessary for mapping and variant calling, and this makes the haplotyping rather more complicated.

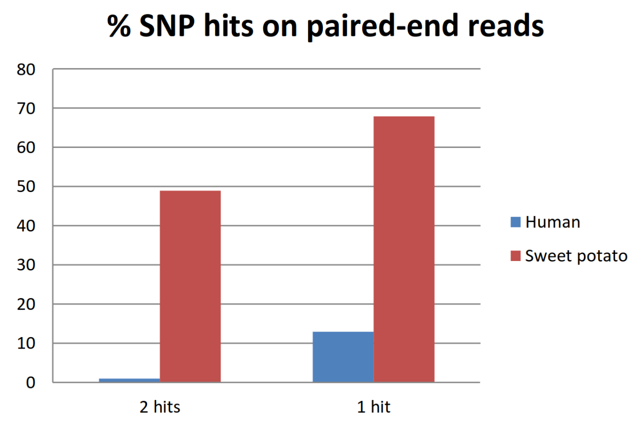


FIGURE 2.8: **The percentage of SmP hits in human and sweet potato genomes.** The percentage of useful reads for haplotyping for human and hexaploid sweet potato is compared. As shown, about 1% and 50% of the reads are usable for haplotyping for human and sweet potato, respectively. The notable difference is due to the heterozygosity of these genomes. The polymorphic sites are much closer in the sweet potato genome such that more read pairs, with 100bp single-end short reads, cover more than two polymorphic sites. The statistics are inferred from an individual in the 1000 genomes project[22] and the pilot sweet potato project data[*Section 6.2*]

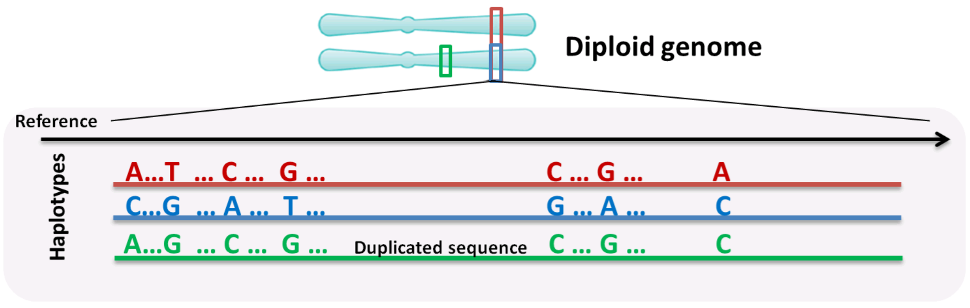


FIGURE 2.9: **Effect of a duplication in haplotyping for the human genome.** The green sequence is a duplication of the blue one. The mapper and haplotype assembler cannot distinguish the reads which are sampled from the green region; consequently, it assumes there are two haplotypes and it merges the reads from three haplotypes to assemble two haplotype sequences.

2.5.4 Repetitive regions

Repeats are defined as similar or identical sequences placed in different positions in the genome. The large genomes contain different amounts of repetitive sequences, for example, human and sweet potato genomes have 50% and 40% repetitive elements, respectively [11, 23]. Repeat elements can be interspersed or tandem. **Interspersed repeats** are elements which are identical or near identical and can be a million base pairs apart in a genome, while **tandem repeat** elements are adjacent. The size of a tandem element can be as small as two base pairs, and the number of the copies in one repeat can be many thousands[24].

Although genome and haplotype assembly could be defined as one problem, *i.e.* obtaining the sequence of all of the chromosomes, they are defined here as two due to the high similarity between autosomal chromosomes and also the limitation in sequencing technologies dealing with the complex regions in the genome such as repeats. These regions cause fragmentation in *de novo* assembly if the reads are not sufficiently long to cross the repeats. *De novo* assembly focuses on the invariant regions in the genome while haplotyping sits on variant sequences with the assumption of having accurate reference and variants. Therefore, here, we discuss the effect of repeats in *de novo* assembly as well since the assembled reference is the input for haplotyping.

The reads sampled from repeats are mapping ambiguously, meaning they can be mapped to several places with a good similarity. Read length plays a major role; if a repeat is covering the read, there is no signal indicating from which repeat sequence the read is sampled. If the read is long enough to span the repeat, or contain part of the flanking regions, the flanking sequences come to help for mapping the read properly. Otherwise, the ambiguous mapping of short reads affects the accuracy of variant calling, and consequently, haplotype assembly. In the case of tandem repeats the difficulty is the estimation of repeat size. *Figure 2.10* shows the effect of repeats in genome assembly (*Figure 2.10-A*), and estimating the size of reference on repetitive sites (*Figure 2.10-B*).

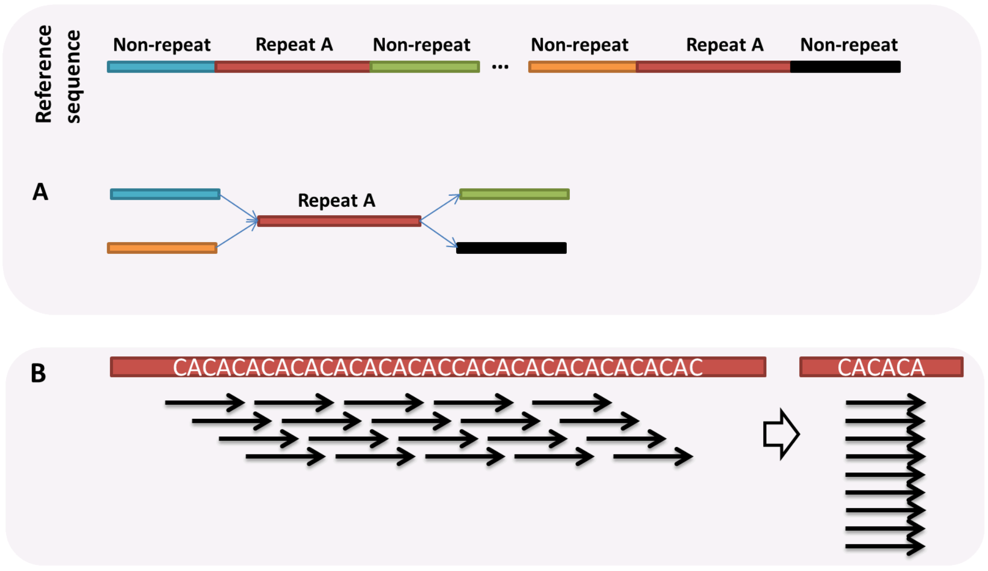


FIGURE 2.10: **Repetitive regions** A) The effect of interspersed repeats in genome assembly. It is possible to find out which sequence (blue or orange) should be connected to the black one. B) The effect of tandem repeats in genome assembly. The assembler might squeeze the region since the reads are short.

2.5.5 Technical issues

Coupled with the genome characteristics, there are a number of sequencing related technical issues which may affect the haplotyping, *e.g.*, read length, error rate, and coverage. For example, the read length of the recruited sequencing technology should be long enough to connect at least two variants. The read length and the genome heterozygosity are two important inter-related characteristics which affect the haplotyping. The other characteristic is the error rate in base calling. With high error rate, the variant caller may not identify the true sequence variants among the erroneous bases. This problem could be addressed by deeper sequencing, which has the downside of higher cost. This becomes more challenging for polyploid genomes. For example, we sequenced the sweet potato genome with one of the earliest MinION Nanopore Flow cells. The low coverage and high error rate meant that we could not distinguish if the alleles in the reads were true or not (*Figure 2.11*).

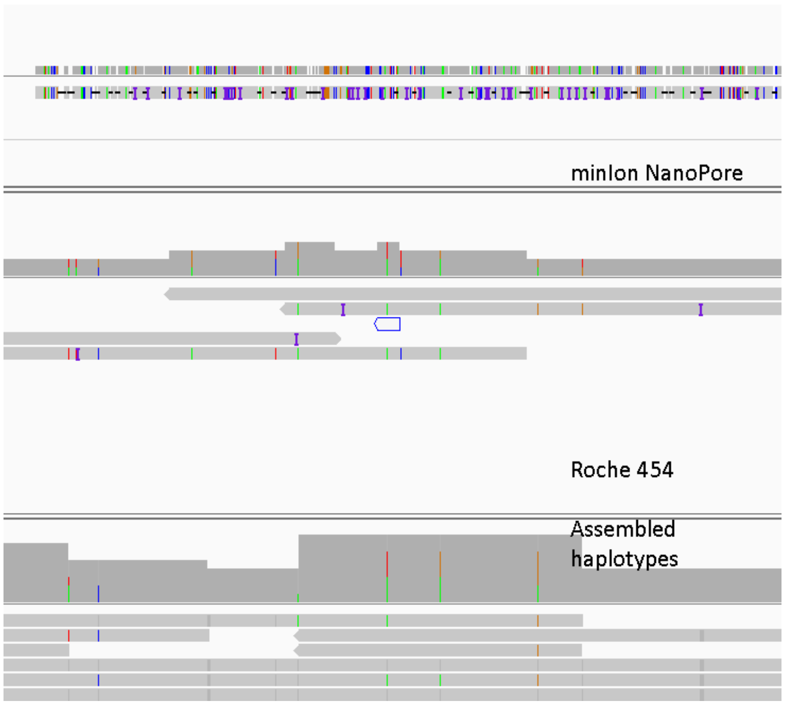


FIGURE 2.11: **An illustration of sequence errors in the genome browser.** The three tracks are MinION Nanopore, Roche 454, and assembled haplotypes from Illumina reads. The dissimilarity between the reads and reference are depicted in different colors. The first track shows the error rate is so high that distinguishing between true variants and error is not trivial.

Moreover, differences between the sequenced individual and the reference and also the errors in the reference may introduce errors in the haplotyping. For re-sequencing data, since the reference sequence is not the same individual as the sequenced one, the structural differences between the individual and reference may prevent assembly of the true haplotype (see *Section 2.5.3)*. Moreover, haplotypes containing SNVs may be more difficult to assemble. One strategy to deal with these errors is *de novo* assembly of the individual together with haplotyping. This approach is costly and needs long reads and high coverage. The currently available sequencing technologies are either rather short, highly accurate, and high-throughput, such as Illumina sequencing, or long, error-prone, and low throughput, *e.g.*, Nanopore and PacBio.

2.6 Ground truth data availability

Obtaining the ground truth data for every organism is a hard task with current sequencing technologies, their costs, and available tools. However, for a decade various groups have addressed human genome haplotyping. The haplotype sequences obtained from genotyping of a trio family released by *The International HapMap Consortium* (phase *II* and *III*, 2007 and 2010 respectively) have served as the ground truth haplotypes for a while. In one dataset, the child and the parents were genotyped, and later on sequenced deeply with Illumina technology for calling SNVs and indels; hence, the child's haplotypes, NA12878, can be considered as the ground truth data. Another set of phased haplotypes, for the human genome, was obtained from the combination of several sequencing technologies, Illumina, PacBio, and BioNano genomics[25].

Such ground truth data, sequenced with different types of sequencing technologies and studied in-depth with the aid of pedigree information, is not available for non-human organisms. Chen *et al.* reported deep PacBio sequencing for diploid *Arabidopsis thaliana*, *Vitis vinifera* (grape), and the coral fungus *Clavicorona pyxidata*. They did *de novo* assembly and haplotyping for these genomes; however, no sequence can serve as the ground truth to evaluate the assembled haplotypes[26]. In Yang *et al.*, we proposed a strategy for evaluation of assembled haplotypes for hexaploid sweet potato genome. It relies on deep Illumina sequencing for haplotype assembly and recruited Roche 454 reads, which were not integrated with the *de novo* assembly, as the ground truth data. Although the Roche 454 reads are limited in size, they are sufficiently accurate for evaluating parts of the assembled haplotype[11].

2.7 Metrics and evaluation

It is important to assess the quality of the computed assembled haplotypes obtained from different methods. Different metrics have been introduced according to the type of input data. Here we list some of these metrics. Generally, these metrics can be divided into two groups based on the availability of ground truth haplotypes. Note that these metrics are applied after the assignment of each assembled haplotype to the ground truth data, if available. Depending on the ploidy of the genome, there could be two or more possible assignments. Each assignment has a score and the following metrics are calculated on the best assignment. *Figure 2.12* shows two possible assignments of computed haplotypes to the ground truth ones. The selected metric is computed for both of the assignments. The assignment with a better score is selected for further analysis. The number of possible assignments in polyploid genome is P! (*i.e.* $P \times (P-1) \times (P-2) \times ...2 \times 1$). *Figure 2.13* shows a triploid assignment as an example. There are six possible assignments each of which is evaluated by its accuracy and the assignment with highest accuracy is used for evaluation.

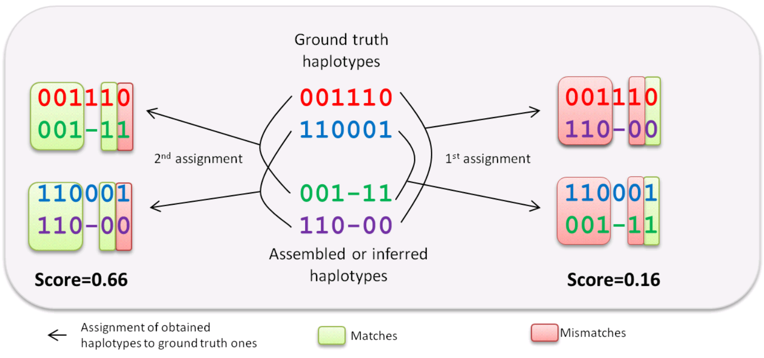


FIGURE 2.12: **The two possible assignments of computed haplotypes to the ground truth ones in a diploid genome.** The ground truth and computed haplotypes are shown in the middle. The two assignments are illustrated as first and second assignments. The selected metric here is the number of matches divided by the length of haplotypes. The second assignment has a better score, or accuracy, and it will be used for the evaluation of computed haplotypes.

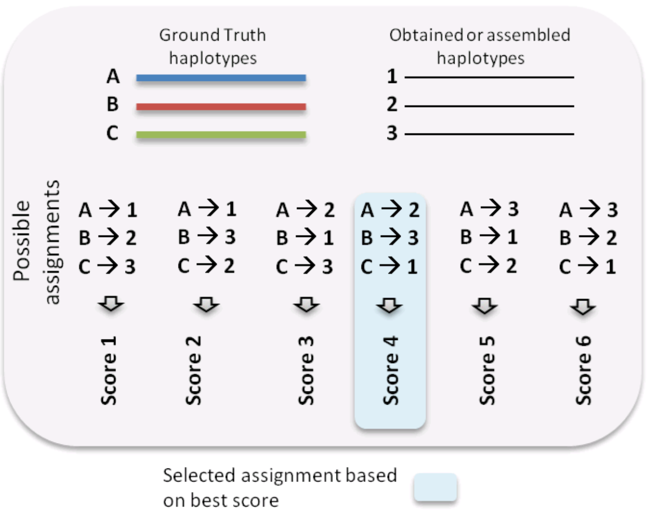


FIGURE 2.13: **The six possible assignments of computed haplotypes to the ground truth ones in a triploid genome.**

- **N50:** This metric is a weighted average length and evaluates the length of the assembled sequences. It indicates half of the assembled sequences are assembled into sequences of this length or longer. *Figure 2.14-A* illustrates this measurement.

- **Variant N50:** It has the same meaning as N50 but instead of base pairs it uses the variants. In other words, the N50 is in nucleotide space and the VN50 is in coded allele space. Variant N50, or VN50, means half of the computed haplotypes in coded allele space are bigger than the VN50 size.

- **Accuracy or reconstruction rate:** It is calculated based on the number of correct matches on heterozygous sites between the assembled haplotypes and the true ones divided by the length of the haplotypes. Two parental haplotypes and two assembled haplotypes are shown in *Figure 2.12* as an example. There are two possibilities for the assignments of assembled haplotypes to the real ones (shown as first and second assignments) with the score of $\frac{4+4}{2\times 6} = 0.66$ and $\frac{1+1}{2\times 6} = 0.16$, respectively; hence, the second assignment is the better one, and the accuracy of 0.66 is chosen.

- **Switch error:** It is the number of allele block exchanges between the assembled haplotypes in such a way that after these exchanges the assembled haplotypes become equal to the true ones. Let's assume that the two haplotypes obtained from a diploid genome are MMPPM and PPMMP, where P and M stand for paternal and maternal alleles. We know the true haplotypes are PPPPP and MMMMM. Therefore, there is a need to switch the assembled haplotype blocks in position two and four to obtain the real ones. So, in this example, the switch error is two. This metric can be applied on diploid genomes since the haplotypes only contain the heterozygous alleles. Moreover, there is just one possible number of switches in diploid genomes. Whenever the ground truth haplotype disagrees with the computed one, a switch is needed. The position of switches in the other haplotype are exactly in the same place (see *Figure 2.14-B*).

  In polyploid genomes there might be several possible sets of switches. *Figure 2.14-C* illustrates two possible set of switches for a triploid genome. Therefore, in this case we need to choose one of all possible sets of switches in order to evaluate the computed haplotypes. In parsimony paradigm, one can select the set with minimum number of switches; hence, **minimum switch error** can be used as a metric for evaluation. However, there is a problem using this measure for polyploid genomes. It might be the case that no possible set of switches can convert the assembled haplotype to the true ones when the detailed genotype data is not available. *Figure 2.14-D* highlights a polymorphic site in which the genotype of ground truth and the computed haplotypes are not identical. Another problem is predicting the frequency of alleles in the polymorphic sites of computed haplotypes. For example, given the genotype data 0,1, and 2 for a hexaploid genome, there are several possibilities for the frequencies of the alleles such as ‘ 0/0/0/1/1/2’, or ‘0/0/0/0/1/2’. In such a situation, no switch error can be computed. One strategy to deal with this problem is to ignore the sites which the frequencies are not matching; however, there might be some correctly assembled haplotypes among all of them which are ignored at the polymorphic sites.

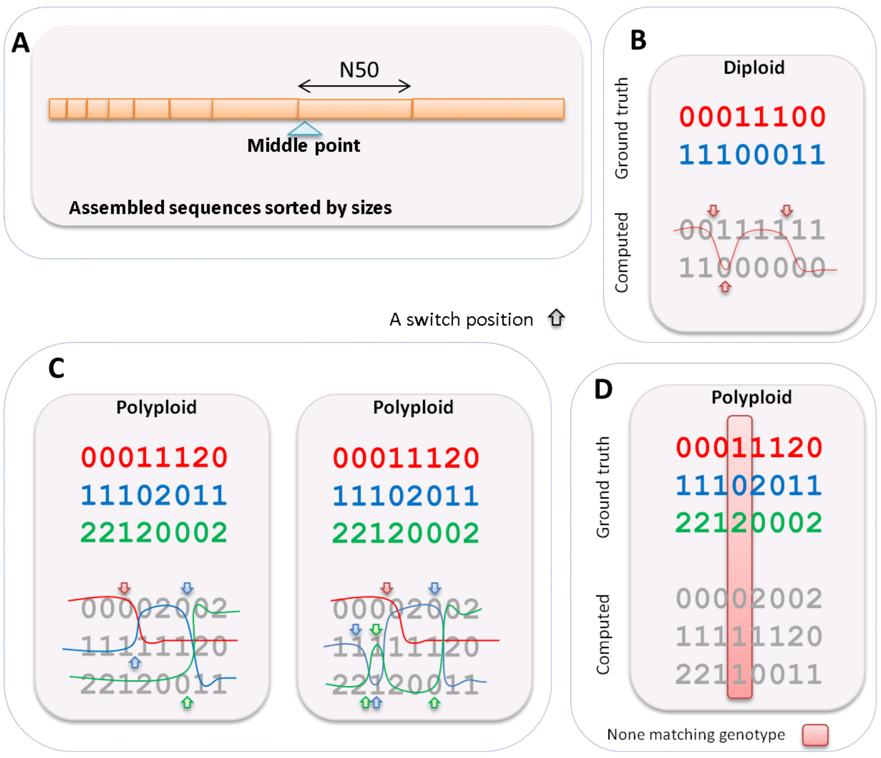


FIGURE 2.14: **N50 and switch error metrics.** Panel A illustrate the N50 measure. Panel B depicts the number of switches needed for converting the computed haplotypes to the ground truth ones. Panel C shows two possible set of switches, namely four and seven switches, for the first and second scenarios, respectively. Panel D indicates the problem of usingutilizing switch error for evaluation of computed haplotypes for polyploid genomes. The highlighted column indicates that if the genotypes of computed haplotypes and ground truth ones at a polymorphic site are not identical, the switch error measure cannot be computed.

- **Completeness** This metric shows how many of the heterozygous alleles are covered by the longest haplotypes. For instance, if one haplotype connects 95 polymorphic sites among one hundred, the completeness is 95%. This measure is defined based on the longest assembled haplotype.

- **Resolution** This metric is also defined for the longest assembled haplotype. It shows how many gaps are present in this haplotype.

- **Error correction:** This metric is based on the minimum error correction model (*Section 3.2*), and it is calculated as the number of corrected errors in assembled haplotype in such a way that all reads assigned to each haplotype are compatible with each other.
- **Weighted error correction:** This metric is calculated in the same way as the **Error correction**, but instead of counting the number of alleles, it sums over the weight of the corrected alleles.
- **Fragment removal:** It is defined based on minimum fragment removal model (*Section 3.2*) and calculates the number of reads needed to be removed to have a non-conflicting partition of reads.

Accuracy is the most widely used metric to evaluate haplotypes. **Switch error** helps mostly when the errors in the haplotypes change the long-range connectivity, such as with using paired-end and mate-pair libraries. **Completeness** and **resolution** allows us to evaluate genome-wide haplotyping when the sequencing technologies are long and accurate enough, and when the sequencing depth is sufficiently high that one haplotype covers most of the genome. **Error correction**, **weighted error correction**, and **fragment removal** metrics provide fitness when the true haplotypes are not available. These metrics check the assignment of reads to the assembled haplotypes and calculate the scores of how well these assignments were done.

Chapter 3

Single individual haplotype reconstruction: techniques and protocols

3.1 Introduction

Haplotyping methods were developed along with advancement in sequencing technologies and the assembly of the human genome. First, second, and third generation sequencing data have been used for haplotyping of a single individual. Although NGS technologies have drastically reduced the price and increased the throughput of sequencing, the length of their reads is too short to connect more than a few of the variants across the genome. However, a number of haplotyping methods were designed to deal with NGS data. These methods have mostly been applied to first generation sequencing data since the underlying principle for the first and second generation, *i.e.*, read accuracy and connectivity with paired reads, are the same. Since then, Illumina sequence reads have become dominant due to the associated low cost, high accuracy, and high-throughput; however, Illumina read length is technically limited and recently reached 2 $\times$ 300bp (Illumina MiSeq series sequencers[27]). To overcome this limitation various strategies have been proposed. Integrating various Illumina insert sizes (*Section 3.4.1)*, utilizing population (*Section 3.4.2*) or RNA-Seq data (*Section 3.4.3*), and recruiting single cell sequencing techniques together with special library preparation for NGS, like *10X Genomics* (*Section 3.5.1)* and Strand-seq (*Section 3.5.2*) are among these strategies. In the next sections, we first explain how single individual haplotype assembly can be modeled, and then, we describe in more detail how the mentioned technologies and protocols can help with haplotyping.

3.2 How to model single individual haplotype assembly

There are several models introduced for haplotype assembly from sequence reads (SIH problem). These models provide a fitness to a set of assembled haplotypes. **Minimum Error Correction** (MEC), **Minimum Fragment Removal** (MFR), and **weighted Minimum Error Correction** (wMEC) are the most well-known models among them. All of these models have been proven to be computationally hard; this necessitates the use of heuristics approaches[28].

Minimum Error Correction (MEC): The most well-known model for SIH problem is the MEC model. It assigns a fitness to a partition and its supporting fragments. For a diploid genome ($P = 2$), according to minimum error correction model, we are looking for two clusters of reads. Each cluster is a collection of reads supporting one haplotype. The haplotype is computed based on majority of votes at each column (polymorphic sites) of the reads which are assigned to the corresponding cluster. MEC model enumerates the number of mismatches between reads and computed haplotypes. The final MEC score is the sum of MEC scores in both clusters. *Figure 3.1* illustrate how the MEC score is calculated for a specific partitioning. There are 2^{n-1} partitions, where n is the number of reads, and the aim of this model is to find the one with minimum number of errors. These errors are flipped or changed such that all fragments in each cluster become compatible with computed haplotype and with each other. The number of allele changes is an indicator of how good the partitioning is. The MEC model suggests that the haplotypes assembled from the partition with the minimum amount of corrected errors are the best ones. It has been shown that haplotyping under the MEC model is an NP-hard problem[28].

Weighted Minimum Error Correction (wMEC): This model integrates the quality of alleles into the MEC model. All alleles in the MEC model are considered to be equally accurate. However, it is possible to integrate the allele quality provided as base calls for single nucleotide variants or the average of base qualities for indels. The goal of wMEC model is finding a partition such that the weights of alleles which are needed to be flipped to have a non-conflicting clusters is minimum. Like MEC, wMEC is NP-hard[29].

Minimum Fragment Removal (MFR and wMFR): This model assumes that the conflicting alleles among the reads, which are assigned to one haplotype, are introduced owing to misalignment. This model sets its objective according to the number of fragments that need to be removed to have non-conflicting clusters. It tries to find the minimum number of such fragments. *Figure 3.1* shows which four reads need to be removed for the illustrated clusters. The SIH problem under the MFR setting is an NP-hard problem as well[29]. The **wMFR** model follows the same principle but each

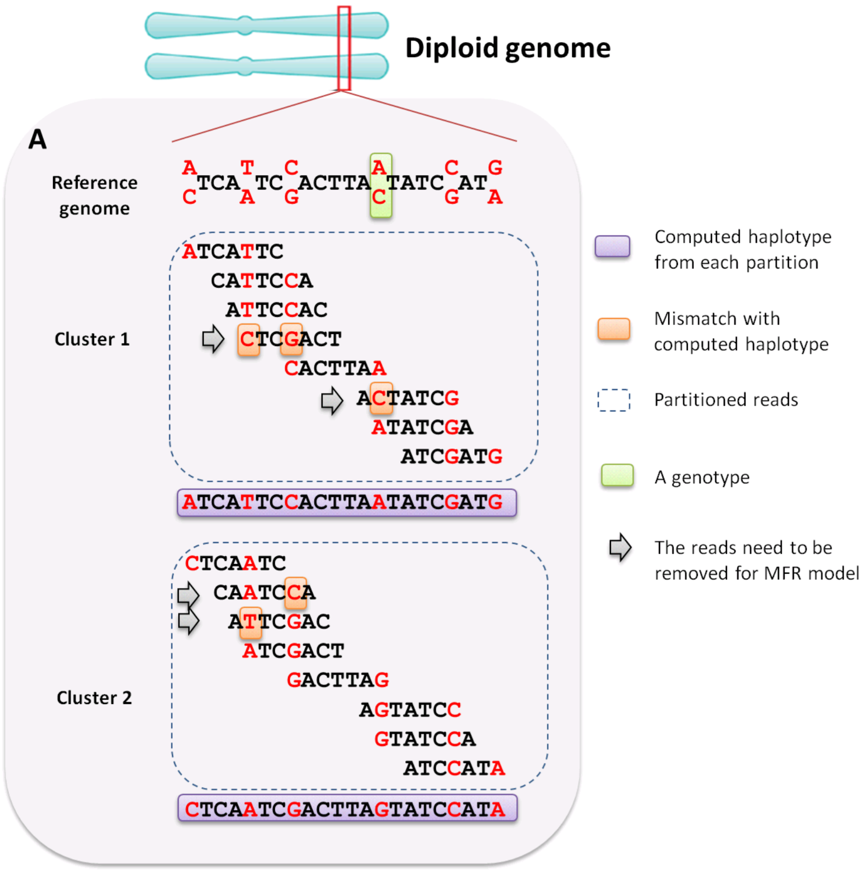


FIGURE 3.1: **An illustration for minimum error correction(MEC) and minimum fragment removal(MFR) models.** For a diploid genome the reads are partitioned into two clusters (cluster 1 and cluster 2). For each cluster one haplotype is computed based one majority of votes on each column (colored in purple). Afterwards, the number of mismatches (in orange) between reads in the clusters and the computed haplotypes are counted. In this example, there are three mismatches in cluster 1 and two mismatches in cluster 2, so in total the score for this partition is five. In MEC model the number of mismatches is an indicator of goodness of partitioning. In MFR model, the goodness of the partitions are calculated based on the number of fragments need to be removed in order to have clusters with no disagreement. In the partition illustrated in this figure, there would be no dissimilarities between the reads in the clusters if the four reads which are pointed out with the arrows are removed.

fragment has a weight and the model calculates the sum of the weights instead of the number of fragments.

The MEC and wMEC models assume that the reads are correctly aligned, and all of the reads should be considered in haplotype assembly. After partitioning, the haplotypes can be assembled via the majority of votes within each cluster. The MFR model checks if a fragment belongs to any cluster or not; and if it does, it should not be in conflict with fragments of the same class. These models provide objective fitness functions to optimize. MEC and wMEC may involve the fragments which are mistakenly mapped to the reference; which is common, especially in repeats. The MFR assumes that errors occur when there is a misalignment, and it eliminates such alignment. This model may remove many informative fragments from error-prone third-generation sequencing data. All in all, although all models are useful as an objective function, none of them are applicable for all types of input data. The best strategy is recruiting a model or combination of them based on the properties of sequencing technology.

3.3 Early single individual haplotyping with sequence reads

The first attempts at assembling the haplotypes of the human genome, which showed the feasibility of haplotyping with sequence reads, resulted in a single individual assembled genome reference called HuRef. To assemble the HuRef, 32 million Sanger sequence reads of 500bp and 800bp lengths were used; these reads were paired by 2, 10, 40, and 100kb clone insert lengths. After *de novo* assembly of the genome with Celera assembler[30], assembled sequences were mapped to the reference genome and the variants were called afterwards. These variants were then identified in the reads and grouped into paternal and maternal reads. Finally, the consensus sequences of each group were represented as the computed haplotypes. This algorithm is explained in more detail in the next paragraph and *Algorithm 1* [31].

Algorithm 1 represents the pseudocode of this approach. A list of aligned fragments (please find the definition of fragment in *Section 2.2)* is considered as the input data (represented as F). The grouping step is done in a greedy manner. First, the fragment containing the maximum number of polymorphic sites is selected and assigned as the initial haplotype (C_1). Recall that in diploid genomes one haplotype is representative of both paternal and maternal haplotypes such that it itself represents one of the parental haplotypes and its bitwise complement represents the other one. Having found the longest fragment (C_1) and its bitwise complement (C_2) as the initial haplotypes, all fragments are clustered into two groups (F^{C_1} and F^{C_2}) according to their similarity to initial haplotypes. This similarity or score is represented as scr_1 and scr_2. Given a fragment f_i, scr_1 (scr_2) indicates the similarity score to C_1 (C_2). f_i is assigned to F^{C_1} if $scr_1 > scr_2$ and is assigned to F^{C_2} otherwise. Then, the consensus sequence of both clusters (F^{C_1} and F^{C_2}) are computed as the new haplotypes (updating C_1 and C_2). This algorithm is done iteratively by assigning fragments to new haplotypes and updating the consensus sequences. The algorithm stops when the haplotypes' sequences do not change in two consecutive iterations.

```
Levy et al. algorithm
input  : a set of fragments $F$
output: a set of haplotypes $H$
$C_1 \leftarrow$ Find a fragment which covers maximum number of alleles
$C_2 \leftarrow$ bitwise complement of $C_1$
$C_1^{old}, C_2^{old} \leftarrow \emptyset, \emptyset$
while $C_1 \neq C_1^{old}$ or $C_2 \neq C_2^{old}$ do
    $F^{C_1}, F^{C_2} \leftarrow \emptyset$
    $C_1^{old}, C_2^{old} \leftarrow C_1, C_2$
    for $f_i$ in $F$ do
        $scr_1 \leftarrow$compute the similarity score of $f_i$ and $C_1$
        $scr_2 \leftarrow$compute the similarity score of $f_i$ and $C_2$
        if $scr_1 > scr_2$ then
            $F^{C_1} \leftarrow f_i$
        else
            $F^{C_2} \leftarrow f_i$
    end
    $C_1 \leftarrow$ consensus sequence of $F^{C_1}$
    $C_2 \leftarrow$ consensus sequence of $F^{C_2}$
end
```

Algorithm 1: Levy *et al.* algorithm

3.3.1 *HapCut*

HapCut[32] is a graph based algorithm which builds a read-haplotype consistency graph. This graph is constructed based on an initial haplotype and the input list of fragments. It starts with a random initialization of one haplotype. As mentioned before, the other haplotype is its bitwise complement. Nodes are the polymorphic sites, so the number of nodes is n if the haplotype length is n in coded allele space. Two nodes are connected if there is at least one read covering their corresponding polymorphic sites. For instance, '101' as an aligned fragment starting at the position one (positions are 1, 2, and 3) contributes three edges to the graph namely, '(1,2)', '(2,3)' and '(1,3)'. The weight of an edge indicates the strongness of co-occurring of variants at the two polymorphic sites in the initial haplotypes and is calculated as the number of fragments between two nodes which are consistent with the initial haplotype minus those which are inconsistent. That is why the graph is called read-haplotype consistency graph. If the read is consistent to the haplotype it contributes positive weight and if it is inconsistent, contributes negative weights. *Figure 3.2* shows an initial haplotype, a set of fragments, and their corresponding graph.

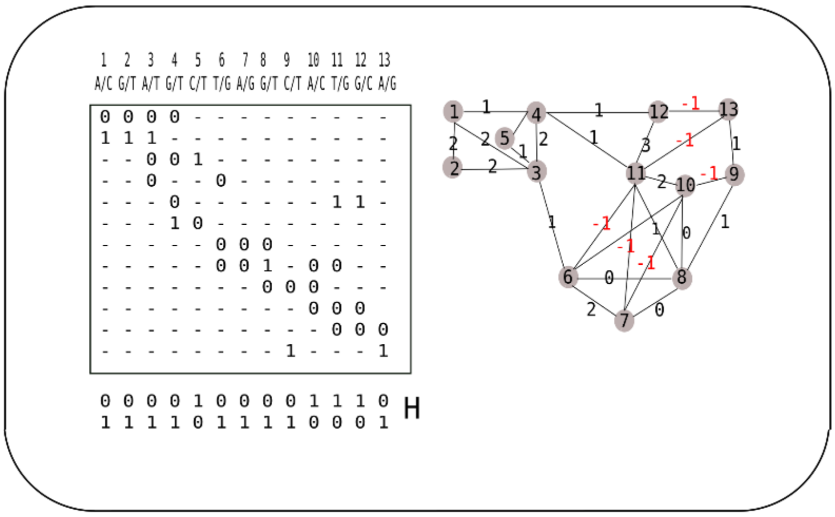


FIGURE 3.2: **Schematic view of *HapCut* algorithm.** Left) A list of fragments and an initial haplotype. Right) Constructed graph based on the initial haplotype and the list of fragment. The weights are calculated as the number of reads supporting the haplotype subtracted by the reads which are inconsistent with it. The red weights are the negative ones. Figure is adapted from [32]

A cut S in the graph is a subset of nodes; so the nodes of the graph is divided into two sets, namely the cut (S) and its complement (S'). Weight of a cut is calculated as the sum of weights connecting S and S'. As mentioned, the weights are obtained based on the initial haplotype, therefore, by bitwise complementing the variants corresponding to S, the following conditions hold:

- The weights of edges within S and also within S' stays similar.
- The weight of edges between S and S' are inverted.

Bansal and Bafna proved that any cut with minimum weight smaller than zero leads to a haplotype with lower MEC score. Therefore, an optimum cut leads to an optimal MEC score. Since finding an optimal cut is NP-hard[33], Bansal and Bafna proposed a greedy heuristics method to find a sub-solution cut and then update the haplotype with respect to the cut. This procedure happens iteratively until no better haplotype can be found. As an example, one good cut in the example shown in *Figure 3.2* is $S = \{6, 7, 9, 13\}$. By flipping the variants in the haplotype at these polymorphic sites, all the red scores are converting to positive scores; however, a number of negative score are also generated such as $w_{8,9}$ becomes '-1' rather than '1'. These negative edges are targeted in the next iterations until no further improvement can be made.

```
HapCut algorithm
input  : a set of fragments $F$
output: a set of haplotypes $H$
$C_1 \leftarrow$ Random initial haplotypes
repeat
    Construct the read-haplotype consistency graph ($G_{C_1}$)
        nodes: SNPs
        edges: Number of consistent fragments − number of inconsistent
    fragments.
    Compute a greedy heuristic cut $S$
    $C'_1 \leftarrow$ Flip the value of $C_1$ at positions $s$ where $s \in S$
    if $MEC(C_1) > MEC(C'1)$ then
        $C_1 \leftarrow C'_1$
    end
until no improvement in $MEC(C_1)$;
```

Algorithm 2: ***HapCut*** algorithm

3.3.2 *ReFHap*

Like ***HapCut***, ***ReFHap*** is also a graph-based haplotype reconstruction method based on MEC model which was proposed by Duitama *et al.*[34]. Unlike ***HapCut*** in which nodes are polymorphic sites, and edges are inferred from fragments, ***ReFHap*** usesutilizes fragments as nodes and the edges are inferred from the overlap between fragments. ***ReFHap*** constructs a graph data structure called conflict graph, meaning that the weight of an edge indicates how likely the two nodes (fragments) are from different haplotypes. High weight of an edge suggests that corresponding fragments of the two nodes are more likely to be from different haplotypes. Weights are calculated based on the number of inconsistent and consistent shared alleles between two fragments. ***ReFHap*** utilizes conflict graph data structure in order to cluster the nodes into two parts. In other words, ***ReFHap*** finds a cut to partition the fragments. In ideal conditions with no error in the input, the graph would be bipartite with no edge within each partition, so the fragments stemming from each haplotype are clustered together. However, due to error there may be edges which violate the ideal scenario. An error could be a miscalling variants which may introduce edges within clusters. This may draw a conflicting edge between the fragments originating from one haplotype. Misalignment may cause loss of an edge between the fragments of different haplotypes.

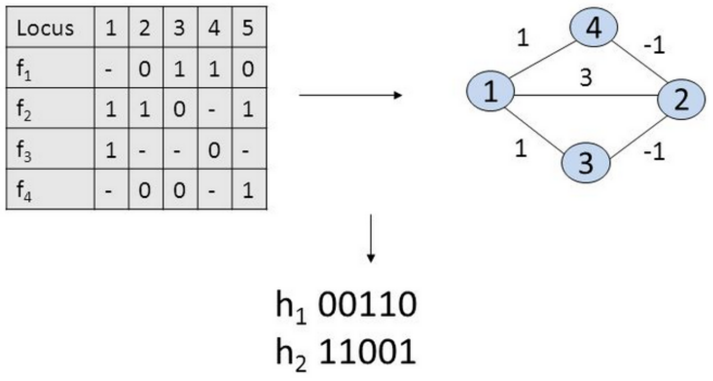
Finding the best cut is NP-hard[33], therefore, ***ReFHap*** follows a heuristic idea in order to find a good cut and more precise haplotypes. ***ReFHap*** starts by assigning scores between all pairs of fragments. It finds a random cut and calculates the MEC score for it. This cut gives an initial score which is obtained randomly and needs to be improved. Then ***ReFHap*** tries to find a better cut and improves its MEC score. For this purpose, ***ReFHap*** sorts the edges in descending order based on their weight. Then, it iterates over the edges ($e \in E_{Sorted}$) in the sorted list.

At each iteration, it assigns the nodes corresponding to the edge ($e = (v_i, v_j)$) into two clusters ($v_i \in S \; and \; v_j \in S'$). Then, all of the other nodes ($e' \in E - e$) are partitioned according to the score of being assigned to each cluster(S or S'). This assignment is done such that the sum of edges inside each group becomes minimum meaning there is less conflict in both clusters. At the end of this step all nodes are assigned to either S or S'. To optimize the cut locally, ***ReFHap*** performs the following steps in the iteration as well. ***ReFHap*** iterates over the edges crossing the two clusters and checks if swapping the nodes improves the overall score or not. It swaps the edges if the swapping decreases the conflicting score. Afterwards, the haplotypes are constructed based on majority of votes in both of the clusters. The constructed haplotypes should be bitwise complementary since the fragments contain only heterozygous variants. In the last step, the sites with homozygous variants can be corrected with the help of fragments supporting the variants. Finally the MEC score for this iteration is calculated and kept if the overall score improves.

In order to improve speed by compromising the accuracy, ***ReFHap*** defines a parameter k for the number of iterations over the reads. It is claimed that $k = \sqrt{|E|}$ could be a good bound in order to achieve a good accuracy. *Algorithm 3* details the steps of ***ReFHap***.

3.4 NGS technologies

Since 2005, the massively parallel sequencing technologies drastically reduced the cost of whole-genome sequencing (WGS) and re-sequencing. Along with the advancement in these technologies, haplotyping methods were developed to use such high-throughput data. Since Illumina sequencers have been the most widely used in the past years, we investigate the methods dealing with such data in the following sections. Although NGS is the most cost-effective way for WGS of the human genome, it is infeasible to link distant alleles or to assemble long stretches of haplotypes with the current read length (100 to 300bp). As shown in *Figure 2.8,* fewer than 1% of reads are useful for human haplotyping with the single-end read of length 100bp. Integrating paired reads



Locus	1	2	3	4	5
f_1	-	0	1	1	0
f_2	1	1	0	-	1
f_3	1	-	-	0	-
f_4	-	0	0	-	1

FIGURE 3.3: **A conflict graph.** This figure shows a conflict graph obtained from the fragment list shown in the upper left part. The fragment list is converted to a conflict graph shown in the upper right. Nodes are the fragments and edges are calculated based on the number of mismatches minus the number of matches between two fragments. A good cut splits fragments into two separate groups $S = \{1\}$ and $S' = \{2, 3, 4\}$. Figure is adapted from [35]

information helps to improve the results, especially using diverse insert lengths and high coverage data, but it remains difficult to expand haplotypes to long DNA stretches using only Illumina reads. Moreover, population, transcriptome, Strand-seq, or Hi-C data can also be integrated with NGS data to boost the performance of haplotypers.

3.4.1 Integration of variable insert length NGS data

Various insert lengths of paired-end reads help to obtain longer haplotypes from short reads. Through simulation, Tewhey *et al.* [36] reported that the integration of different insert sizes and production of sufficient coverage compensate the short length of Illumina reads. *Figure 3.4* indicates that ∼100 fold coverage of three inserts results in the same haplotype variant N50 as a read length of 10kb with 50 fold coverage. In this analysis, the size of a single-end read is 100bp; the paired reads are sampled randomly from chromosome 1 of a Yoruban individual (NA19240 from the 1000 genome project), and the insert length distributions are normal with the mean of 2, 5, and 10kb and standard deviation of 10%. The same variant N50 could be obtained by only using 10kb mate-pairs but higher coverage (∼140 folds).

3.4.2 Integration of NGS and population data

Yang *et al.*[37] proposed a method, called haplotyping with reference and sequence technology, shortly HARSH, which combines haplotype assembly and haplotype phasing to obtain more accurate haplotypes. Since thousands of individuals were and more are going to be sequenced, utilizing reference panels of haplotypes can be very useful for single

```
ReFHap algorithm
input  : a set of fragments $F$
output: a set of haplotypes $H$
Construct the Fragment graph ($G(V, E)$)
    $V = f \in F$
    $E$ = number of inconsistent alleles between two fragments − number of consistent ones
$C \leftarrow$ Random initial cut
$E_{Sorted} \leftarrow$ Sort edges in descending order
$E^k_{Sorted} \leftarrow$ Select the $k = \sqrt{|F|}$ highest values
for $e = (f_i, f_j) \in E^k_{Sorted}$ do
    assign $f_i$ and $f_j$ to two different partition $S_1$ and $S'_2$
    repeat
        $(C_i, v) \leftarrow$find $v$ such that by assigning it to either $\hat{C}_1$ or $\hat{C}_2$, sum of the scores becomes maximized
    until $\hat{C}_1 + \hat{C}_2 \neq V$;
    Two local optimizations by flipping edges and nodes to see if it boosts the score or not
    if $score(\hat{C}_1, \hat{C}_2) > score(C_1, C_2)$ then
        $(C_1, C_2) \leftarrow (\hat{C}_1, \hat{C}_2)$
    end
end
```

Algorithm 3: ***ReFHap*** algorithm

individual haplotyping. In other words, the reference haplotype panels are getting more accurate over time; therefore, the assembled haplotypes are getting more accurate as well. From the sequence data, the suggested probabilistic model tries to find two haplotypes with the minimum number of conflicting alleles with reads data (MEC model). From the reference haplotype panel data, it utilizes the statistical model for patterns in linkage disequilibrium suggested in [38]. In general, the aim is to obtain haplotypes that are consistent with both sequence reads and reference haplotype panel.

3.4.3 Integration of NGS and transcriptome data

Missing the connectivity of distant alleles by paired reads is the major problem for haplotyping with merely genomic sequence data. Transcriptome data can help the phasing of the exonic regions with short reads. This idea is motivated by two factors:

- RNA-Seq read pairs are both sampled from one chromosome. They provide information about the haplotype which the gene was transcribed from. Since each read (or part of a read) of a transcript is sampled from one mRNA, there is a

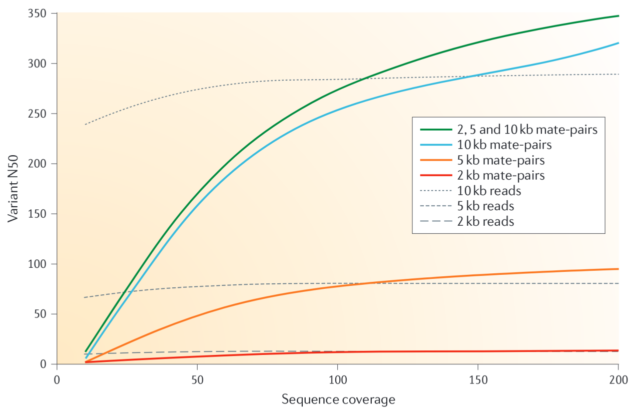


FIGURE 3.4: **The comparison of different insert size on the haplotype assembly of human genome.** The y-axis shows the variant N50 (this figure is adapted from [36]).

chance of sampling one read (or part of it in split reads) from one exon and the other read (or in part) from another exon; which provides longer distance allele connectivity than normal DNA-seq reads. As an example, if two reads of a pair, with an insert of 1kb, are sampled from two consecutive exons, the paired-end read information gives the connectivity of 1kb + intron size (*Figure 3.5-b*). Berger *et al.* suggest this idea to leverage single individual haplotyping[39].

- Differential allele-specific expression (DASE) information within RNA-Seq data can be employed to detect the connectivity of the distant variants. If the expression of the maternal and paternal chromosomes differs significantly, it is possible to assign the exons with the same level of transcription into one haplotype. Differential haplotypic expression gives the information on variants' connections in exonic regions in the range of gene length (*Figure 3.5-c*).

HapTreeX integrates this information into a haplotyper. It works under a probabilistic model; it searches for a collection of partial solutions according to a relative likelihood function of reads and a set of candidate haplotypes. This collection consists of a set of k-length haplotypes, from the 1^{st} SNP to the k^{th} SNP. ***HapTreeX*** extends these haplotypes to the $k+1^{th}$ SNP position. It iterates this procedure until it covers the full haplotype length[40].

3.4.4 Chromosome isolation

One direct, yet complicated and labour intensive, approach to haplotyping is chromosome isolation. Fan *et al.* proposed to separate chromosomes during the metaphase to capture each chromosome. Afterwards, the method collects chromosome by chromosome, makes two sets of parental chromosome sets and then amplifies and genotypes

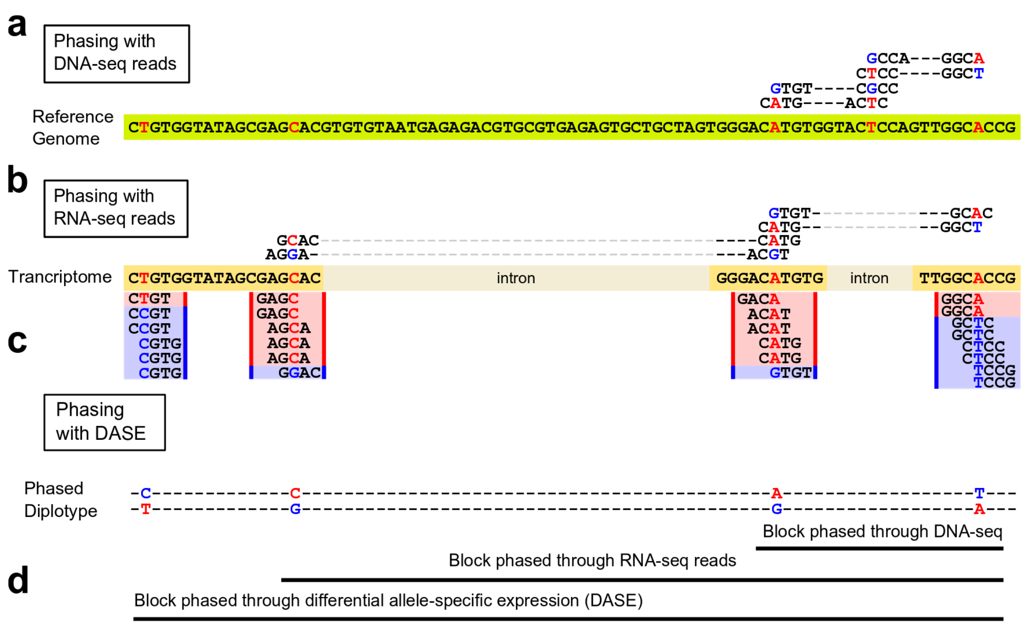

a
Phasing with DNA-seq reads
Reference Genome
b
Phasing with RNA-seq reads
Trancriptome
intron
intron
c
Phasing with DASE
Phased Diplotype
Block phased through DNA-seq
Block phased through RNA-seq reads
d
Block phased through differential allele-specific expression (DASE)

FIGURE 3.5: **Haplotyping with the help of transcriptome data** This figure shows the allele connectivity via paired reads (a), RNA-Seq reads (b), and DASE information (c). By 'DNA-seq reads', here, we mean genomic sequence reads. Part (d) shows the length of the region of effect for each of the information layers. This figure is adapted from [40]

each set. Since the alleles in each set belong to one chromosome, there is no need for haplotype assembly. In principle, it is possible to sequence each chromosome set to detect SNVs as well[41].

3.5 Single cell sequencing technologies with NGS

3.5.1 Linked-read data (*10X Genomics*)

Zheng *et al.* introduce another type of sequence reads called **linked-reads**[42]. Conceptually, a linked-read is a collection of highly accurate short reads which are known to be sampled sparsely from one chromosome. This information is provided by means of sequence barcodes; however, the order of the reads in the fragment is not known but can be inferred, for example, by mapping them to the reference. The underlying technology works based on "in emulsion PCR". It consists of 1) a number of gel beads, each of which contains 16 letter unique barcodes, necessary sequence adapter and polymerases and 2) a microfluidic system. High-weight DNA sequences (<100kb) are loaded into the gel-beads (or droplet) and the fragments in each droplet are barcoded with one barcode. Afterwards, the emulsion is broken and the reads with the barcodes are sequenced (*Figure 3.6*). Since both the number of fragments per gel-bead and the size of the high-weight DNA sequences in each gel-bead are limited, the chance of loading two fragments from two chromosomes of the same homolog is very low (0.5%[42]). The length of initial fragments can also be integrated to reduce this chance. Therefore, it is highly probable that the reads with the same barcode that are mapped to a specific region in the genome are from one haplotype. Although the coverage of each fragment in one droplet is very low, it provides highly accurate information about distant alleles.

***Long Ranger*:** This software integrates linked-reads and defines a probabilistic model consisting of the matrix of connected alleles via linked-reads together with the variant qualities as the observation, and it tries to maximize the likelihood function of having a phase given the observed reads. It is worth mentioning, in order to further reduce the chance of seeing two fragments from different homologs which by chance are loaded to one droplet, those barcodes with inter-read gap lengths of >50kb are excluded. The algorithm, firstly, tiles the genome into blocks of 40 variants and applies a beam search algorithm to find a near-optimal solution. Then the junctions between blocks are connected via the reads spanning the consecutive blocks. Afterwards, it inverts the haplotype assignment iteratively to increase the likelihood until it converges. The last step of the algorithm is finding the weak break point to break the haplotypes into highly confident blocks. These breakpoints are obtained by inverting the haplotype on the left side of the breakpoint and calculating the score of the inverted one. This algorithm results in haplotypes of several Mbs[42].

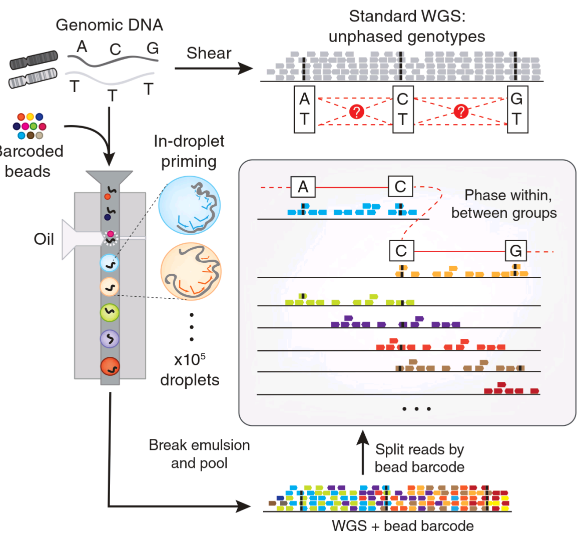


FIGURE 3.6: **The schematic view of 10X genomic pipeline from high molecular weight fragments to assembled haplotypes.** (adapted from [43])

3.5.2 Strand-seq

Strand-seq is a single cell protocol that allows sequencing of a single strand of a chromosome, either forward (Crick strand, shortly C) or reverse (Watson strand, shortly W). The complement strand of each sister chromatid is labelled with bromodeoxyuridine (BrdU) during replication, instead of thymine. After cell division, there are four possibilities for the original strands in daughter cells, *i.e.*, containing C/C, W/W, W/C, and C/W. The replicated strand can be distinguished by the BrdU labels and removed before sequencing. Therefore, after the sequencing of each cell, the original strand of the mother cell can be distinguished by mapping the reads onto the reference. The reads from the Crick (Watson) strand would be mapped to the forward (reverse) reference; hence, the paternal and maternal reads will be separated according to the mappability to the forward/reverse reference. Therefore, W/C and C/W regions are used to give haplotype information.

One difficulty of using such data is the low coverage, which is about 0.1x. Thus, sequencing of one single cell provides very sparse but chromosome-wide haplotype information. The connection between alleles could span the areas problematic for haplotyping such as centromeres and non-heterozygous regions. To deal with the low coverage, many cells need to be processed[44]. This protocol moves the challenge from haplotyping to sequence assembly since the origin of each read can be obtained via the directionality of the mapping.

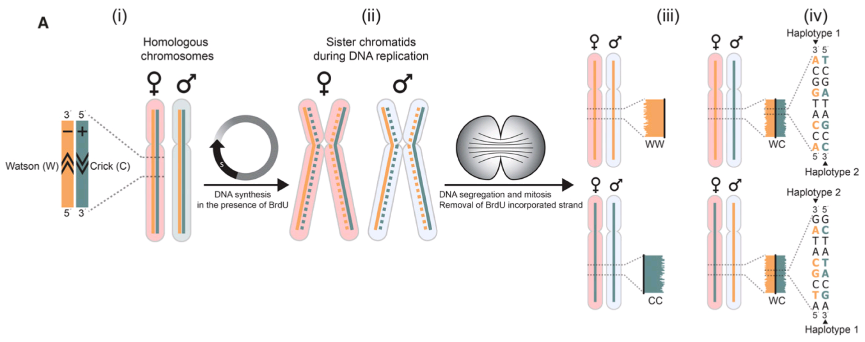


FIGURE 3.7: **Strand-seq pipeline.** (adapted from [44])

3.6 Third-generation sequencing

Like for genome assembly, the accuracy and the length of sequence reads are among the main parameters for haplotype assembly. Third-generation sequencing, which is also called Single Molecule Real-Time (SMRT) sequencing, such as PacBio and Nanopore, provides much longer reads than NGS data with the length ranging from 2-20kb, albeit a higher error rate (10%). These reads can connect several polymorphic sites and improve the allele connectivity for haplotyping. However, the high error rate can convert one allele to the other and cause flipping or switch errors in assembled haplotypes. The high error rate could be reduced by making consensus reads via all-vs-all alignment of these reads; however, the strategy of making consensus sequences comes with the cost of needing higher coverage.

3.6.1 *ProbHap*

Kuleshov suggests a probabilistic model for human haplotyping called ***ProbHap***. He designed a probabilistic dynamic programming algorithm on a probabilistic graphical model which optimizes MEC objective function. This model uses the sum-product message passing algorithm, which is a generalised model for HMM (Hidden Markov Model). The objective function is based on the likelihood of reads' assignment to haplotypes and probability calculation of this assignment given the alleles and their quality. The former are the hidden variables of the model while the latter are considered as the observed variables. [45].

3.6.2 *WhatsHap*

WhatsHap is another approach designed for so called future-generation sequencing methods. This tool is developed to be independent of the size of the reads (speculating that reads will get longer in the future). Like ***ProbHap***, the quality of bases is considered in ***WhatsHap***; hence, by current sequencing technologies it fits best with third-generation sequencing data. The ***WhatsHap*** core is using the coverage as the fixed parameter. However, when the fixed parameter is more than 20x coverage, its performance drops. ***WhatsHap*** usesutilizes a dynamic programming approach to solve a wMEC problem. In the wMEC model the reads are partitioned into two clusters such that the objective function becomes optimized. The score of a partition is computed by the sum of the alleles' scores which needs to be flipped in such a way that the clusters become conflict free. ***WhatsHap*** goes from one end of the scaffold to the other and calculates the score of all possible $2^{|reads|}$ bipartitions to find the one with the highest objective score.

3.6.3 Third-generation sequencing plus NGS reads

Pendleton *et al.* proposed a single human individual *de novo* assembly and haplotyping pipeline. They used 44x coverage of PacBio reads of NA12878 genomic DNA for contig assembly using Celera [46] and Falcon assemblers. The contigs were connected through BioNano genome maps using 80x coverage of long molecules (>180 kb) with mean spans of 277.9 kb[25]. BioNano, in short, is an optical physical mapping strategy which replaces specific sequence motifs with a fluorescently labelled sequence on a HMW, *i.e.* High Molecular Weight fragments (currently up to 2Mb). The fragments are loaded into a chip and are linearised and imaged. Knowing the labelled sites in the fragments and corresponding contigs helps the scaffolding procedure (BioNano Genomics).

For haplotyping, they firstly compared the SNVs and indels called from the high-depth Illumina trio and PacBio sequencing. They found far more heterozygous polymorphic sites with Illumina sequencing than only with the PacBio data. Therefore, they integrated the Illumina and PacBio reads to call the SNVs and then connect them via highly confident reads of both types. They recruited ***HapCut***[32] as the haplotyping method to obtain the final haplotypes[25].

3.7 Haplotype reconstruction for polyploid genomes

Most of the methods for diploid haplotyping, such as ***HapCut***, base their algorithm on the fact that the haplotypes in diploid genomes are the bitwise complement of each other, so they assemble one haplotype and infer the other as its complement. These methods start with an initial haplotype, which is one among $2^{\#heterozygous\ sites}$ possible haplotypes. The initial haplotype can be random or a local optimum based on the input data. However, this property doesn't hold for polyploid genomes. First, all haplotypes should be independently inferred, and the alphabet for haplotypes are not binary; hence, one true haplotype is one possibility among $P^{\#heterozygous\ sites}$ possible haplotypes. This characteristic, along with the ambiguity of merging fragment property (see *Section 2.5.1*), defines a new problem for polyploid haplotyping. Nevertheless, in a broader view, both problems can be considered as a clustering problem with P groups, $P = 2$ for diploid and $P > 2$ for polyploid genomes. In a schematic view, there are different layers of information for polyploid haplotyping, which are depicted in *Figure 3.8*.

There are a number of methods which are designed for polyploid genomes. An extension of ***HapCompass***, which was initially designed for diploid haplotyping, ***SDhaP***, ***H-PoP***, and ***Ranbow*** are the most recent and well-known haplotypers. ***HapTree*** also has an extended version for polyploid genomes, but is excluded here, since it needs genotype information as input, and it has technical issues being run on real and simulated data.

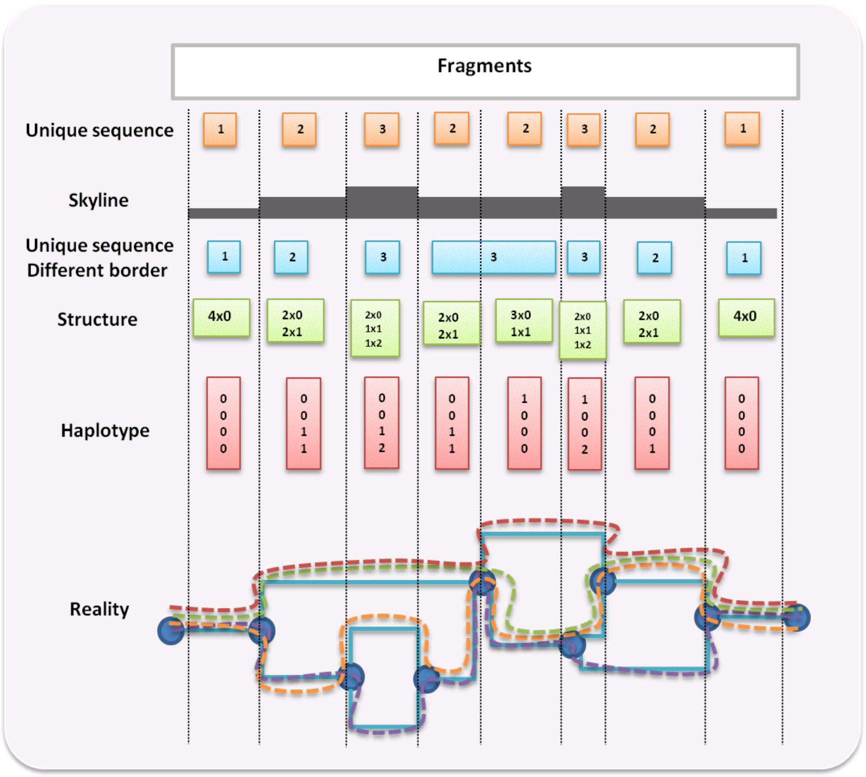


FIGURE 3.8: **Different layers of complexity for polyploid haplotyping. Skyline layer:** The number of unique sequences of chromosomal segments. **Structure layer:** The number of copies of each unique sequence. **Haplotypes layer:** The number of variants which are arranged into the haplotypes.

3.7.1 *HapCompass*

Aguiar and Istrail[47] proposed a graph data structure from reads and polymorphic sites, called compass graph, to solve the SIH problem under wMFR model. The nodes in this graph are the possible phases for every two adjacent polymorphic sites. For instance, for a diploid genome, the two possible phases are ‘00’ (and its complement ‘11’, for simplicity we merge both and use ‘00,11’) and ‘10,01’. The edges are calculated based on the number of reads supporting each combination. For example, if there are 5 reads supporting ‘00’ or ‘11’, and three reads supporting ‘10,01’, the weight is calculated as $5 - 3 = 2$. Using this graph, each path provides a haplotype. Positive edges indicate that ‘00’ is the more pronounced phase, while the negative numbers suggest ‘10,01’ as the phase. The cycles in the graph might be desired (happy cycles) or conflicting. If the number of negative edges in a cycle is even then the cycle is happy, otherwise, it’s conflicting. Aguiar and Istrail defined the problem as removing the minimum number of edges such that the graph becomes conflict-free. Each spanning tree in a happy compass graph gives the haplotypes. Based on this, ***HapCompass*** finds the maximum spanning tree first, and then searches for the conflicting cycles, and solves them until the graph becomes happy (*Algorithm 4*).

This algorithm is designed for diploid genomes and needs to be generalized for polyploid ones. Aguiar and Istrail modified the algorithm and the graph structure for polyploid genomes. Firstly, the weight definition is redefined based on the likelihood of a possible phase owing to the complication rooted in the AoM problem, *i.e.* ambiguity of merging problem (see *Section 2.5.1*). For instance, there are two possibilities for connecting '00,01,10' and '00,01,10' nodes (i.e, '000,010,101' or '001,100,010') in a triploid genome. The weight indicates which one is more probable. Secondly, ***HapCompass*** finds a spanning tree, based on the algorithm mentioned above for diploid, then it keeps the nodes in the order they appear in the spanning tree. This graph with the ordered nodes is called a chain graph. The chain graph can be considered as a network flow with P sources and P sinks. P paths with disjoint nodes and disjoint edges are desired, each of which represents one haplotype[47].

HapCompass algorithm

```
input  : Set of fragments $F$, set of polymorphic sites
output: Set of haplotypes $H$
$G \leftarrow$ Construct the Compass graph
$T \leftarrow$ Find the maximum spanning tree of $G$
$C_G \leftarrow$ Find all conflicting cycles
for $c \in C_G$ do
    Resolving the conflicting cycle $c$ by removing edges
    Updating the compass graph $G$ after edge removal
end
```

Algorithm 4: *HapCompass* algorithm

3.7.2 *SDhaP*

SDhaP uses a semi-definite programming approach that aims to find an approximate solution for haplotype reconstruction for polyploid genomes problem by a greedy search in the space of all possible phasings. The algorithm starts with random initial haplotypes and tries to find solutions with a maximum score by making changes in the initial haplotypes according to a gradient descent method[48].

3.7.3 *H-PoP*

Xie *et al.* proposed a method called ***H-PoP***, which models the problem as an optimal poly-partition problem of reads, called Polyploid Balanced Optimal Partition (PBOP) model. For the P-ploid genome, PBOP clusters the reads into P groups such that it maximizes the defined fitness function. This fitness function is calculated according to the combination of distances between the reads of each group and the distance between the reads of different clusters. Since this model is NP-hard, they designed a heuristic dynamic programming algorithm that considers the solution of m reads and then extends this solution with the aid of the $m + 1^{th}$ read[49].

3.8 Summary

In this chapter, we reviewed the technologies, protocols, and approaches for single individual haplotyping. Firstly, we reviewed human haplotyping and then we mentioned the polyploid haplotype assembly tools as well. Each of the reviewed types of input data has a domain of target polymorphic sites. RNA-Seq data provides distant variant connectivity in the range of gene size and the whole genome, respectively. Population data helps to fill the gaps in the inner recombination hotspot regions. Additionally, the linked-reads give long range connection, in the range of 100kb according to the library preparation, with high accuracy, but with a huge amount of gaps in between. Strand-seq protocol provides the whole chromosome range connectivity but in a very sparse way. Recently, third-generation sequencing technologies, such as PacBio and Nanopore, provide long sequence reads. The drawback of using such data is its high cost and error rate. *Figure 3.9* represents a schematic view of how different methods find the connectivity of variants. All in all, the mentioned methods are either inefficient to call whole chromosome haplotypes or are costly; hence, this field is still open to both methodology and technology improvement.

In the last section of this chapter, we reviewed the state-of-the-art methods for polyploid haplotyping. We explained why polyploid haplotyping is more challenging than diploid haplotyping. Moreover, some of the mentioned protocols for diploid haplotyping are either too expensive and laborious to be applied to polyploid genomes, such as haplotyping with chromosome isolation, or not applicable owing to the higher ploidy, like Strand-seq. High error rate in third-generation sequencing data makes the problem more complicated since it is hard to distinguish true alleles from erroneous sequences. However, the high heterozygosity which is seen more often in polyploid genomes helps to utilize the highly accurate short reads for phasing; the higher the heterozygosity, the greater the chance of capturing alleles through short sequence reads. This characteristic led us to design a novel method for polyploid haplotyping. In the next chapter, we explain this method in detail.

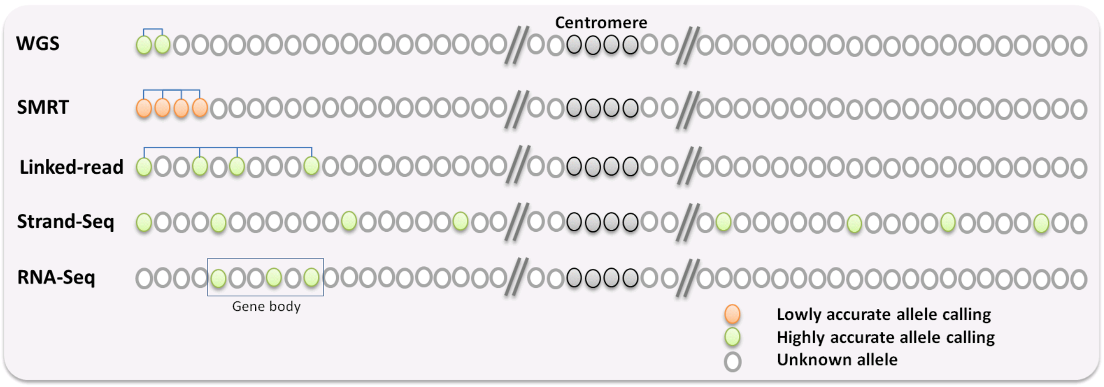


FIGURE 3.9: **A schematic view of allele connectivity via different types of data.** One circle is one variant, either called (green and orange) or not called; The called variants could be lowly accurate (orange) or highly accurate (green)

Chapter 4

Haplotype assembly of polyploid genomes

4.1 Introduction

In the previous chapter, the technologies, protocols, and methods to address single individual haplotyping for diploid and polyploid genomes were reviewed. In this chapter, we explain a novel method, called ***Ranbow***, for reconstructing haplotypes of a polyploid genome from sequence reads (the name is inspired by “rainbow”; as raindrops split the white light into the light spectrum, ***Ranbow*** splits the reference sequence into its haplotypes). In the following sections, firstly, we give the mathematical problem definition of polyploid haplotype assembly. Then, ***Ranbow*** is explained in detail. Note that the terms such as interpolymorphic regions, haplotypes in nucleotide space, haplotypes in coded allele space, which are used in following sections, are defined in *Section 2.2*

Recall that for haplotype reconstruction of an organism, its reference sequence (Fasta file), variants (VCF file), and aligned reads (BAM or SAM file) are consider as the input. The goal is producing a list of aligned sequences as the haplotypes.

4.2 Problem definition and formulation

A haplotype is the sequence of nucleotides in one chromosome. The homologous chromosomes of an individual are similar to each other, except for the polymorphic sites; thus, these haplotypes could also be considered as series of interpolymorphic regions, regions which are equal to each other among homologous chromosomes, (denoted as d_i's) and the heterozygous variants at SmP sites, *i.e.* Small Polymorphisms, (shown as $v_j[i]$'s)

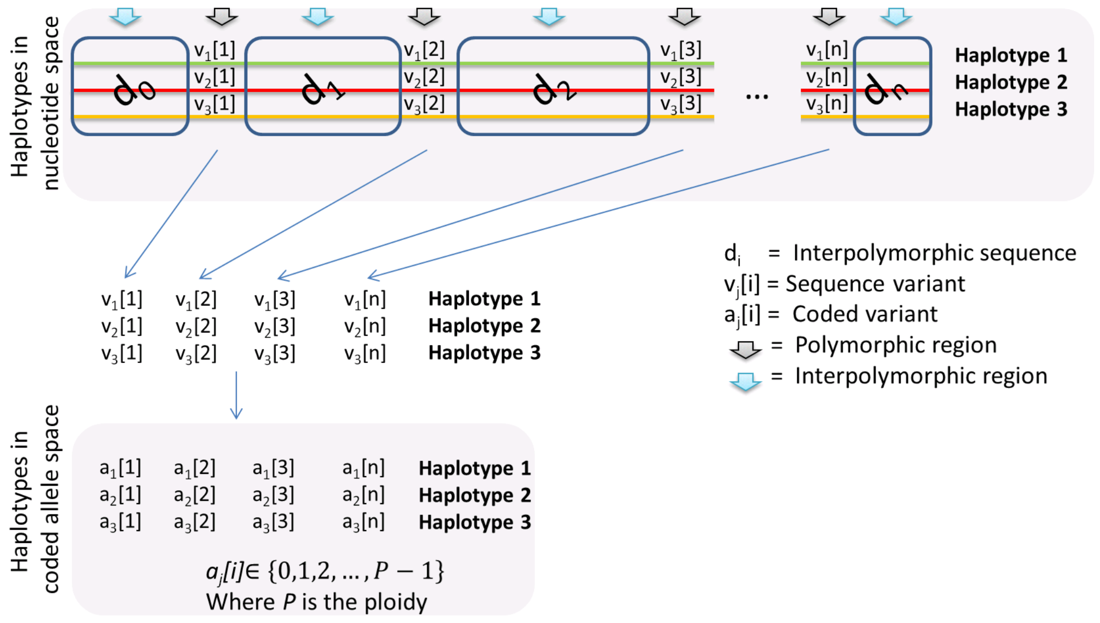


FIGURE 4.1: **Haplotypes in nucleotide and coded allele space.** This figure illustrates how haplotypes are transferred from nucleotide space to coded allele space.

between them (see *Figure 4.1-nucleotide space panel* for illustration and *Section 2.2* for the definitions). For example the haplotype sequences in a triploid genome with three polymorphic sites are:

$$\begin{aligned} h_1^{\text{Nucleotide space}} &= d_0.v_1[1].d_1.v_1[2].d_2.v_1[3].d_3 \\ h_2^{\text{Nucleotide space}} &= d_0.v_2[1].d_1.v_2[2].d_2.v_2[3].d_3 \\ h_3^{\text{Nucleotide space}} &= d_0.v_3[1].d_1.v_3[2].d_2.v_3[3].d_3 \end{aligned} \tag{4.1}$$

where ‘.’ indicates the concatenation operation of sequences. Due to the fact that d_i’s are identical among all homologous chromosomes, they can be removed and a haplotype can be simplified and represented as (see *Figure 4.1*):

$$\hat{h_j} = v_j[1].v_j[2].v_j[3]...v_j[n] \tag{4.2}$$

where n is the number of polymorphic sites and $v_j[i]$’s are the variants. For more simplicity $v_j[i]$’s are coded into numbers ranging from zero to $P-1$, where P is the ploidy of the organism, 0 refers to the reference allele, 1 refers to the first alternative allele, 2 to the second one and so on. By this definition, all type of SmPs are coded into numbers. Recall that a SmP could be a single nucleotide polymorphism, multinucleotide polymorphism, or an indel (see *Section 2.2* for definitions). Therefore, a haplotype can be written as *Eq.4.3*, which is called haplotype in coded allele space. (see *Figure 4.1-Coded allele space panel* as an example):

$$h_j^{\text{Coded allele space}} = h_j = a_j[1].a_j[2].a_j[3]...a_j[n] \tag{4.3}$$

where $a_j[i]$'s are the coded variants. In this chapter, by default haplotypes are considered in coded allele space.

In order to convert back a haplotype from coded allele space into nucleotide space after haplotype assembly, first the numbers are transformed to variants ($a_j[i] \rightarrow v_j[i]$) and then, the interpolymorphic sequences (d_i) are inserted between the variants.

Computed versus ground truth haplotypes: In order to clarify if a haplotype is computed or is a ground truth, we define the following sets, where H is a set of computed haplotypes and H^{True} is a set of ground truth haplotypes:

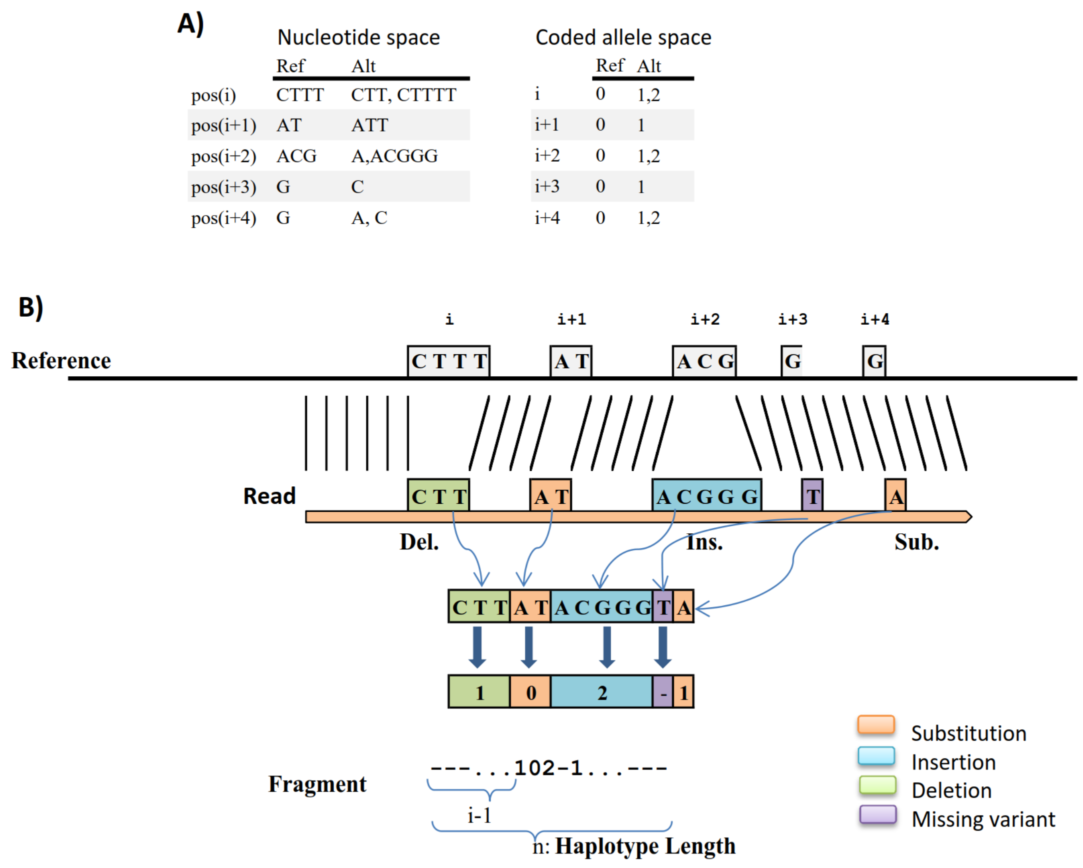
$$\begin{gathered} H^{True} = \{h_1^{true}, h_2^{true}, h_3^{true}, ..., h_P^{true}\} \\ H = \{h_1, h_2, h_3, ..., h_q\} \\ \text{where } q \leq P \end{gathered} \tag{4.4}$$

where P is the ploidy and q is the number of assembled haplotype. q might be less than P because there might be a lack of signal such as missing sequence reads from haplotypes.

Fragment: A fragment f is a sequence of variants which are covered by an aligned read (recall that the variants and the reads are the inputs). To construct a fragment from a read, firstly variants are found in the read. Then, the regions between polymorphic sites are removed and each variant is coded into a number according to the genotype. *Figure 4.2* depicts how a read is converted to a fragment. *Figure 4.2-A* shows sequence variants and how they are coded into numbers. *Figure 4.2-A* shows two tables, one containing a list of variants and their positions in nucleotide space and the other containing the same variants but in coded allele space. The former helps to identify the starting point, the length, and the sequence of a variant, while the latter transforms them into coded allele space.

Figure 4.2-B illustrates a read which covers four sequence variants and a missing allele (explained in the next paragraph): a deletion, a reference variant, an insertion, a missing allele, and a substitution, respectively. These variants are the ith to i+4th variants in the variant list meaning that the variants before ith and after i+4th positions are not covered by the read.

If a read contains a variant which is not reported in the list, it is considered as missing data or a **missing allele** (shown in *Figure 4.2-B* in purple). In general, a missing allele in a fragment is a SmP, or small variant, which is not covered by the fragment and is represented by '−'. If the fragment contains a variant, we need to checked whether it is reference or alternative and how it is coded into numbers (*Figure 4.2-A-right*).



A)

	Nucleotide space		Coded allele space		
	Ref	Alt		Ref	Alt
pos(i)	CTTT	CTT, CTTTT	i	0	1,2
pos(i+1)	AT	ATT	i+1	0	1
pos(i+2)	ACG	A,ACGGG	i+2	0	1,2
pos(i+3)	G	C	i+3	0	1
pos(i+4)	G	A, C	i+4	0	1,2

FIGURE 4.2: **Conversion of a read to a fragment:** There are three type of variation (insertion, deletion, and a substitution) covered by the read. This figure shows how the information is formulated into the fragment sequence.

Similarity and dissimilarity functions: Similarity and dissimilarity functions are defined as follows:

$$\begin{aligned} sim(f_1, f_2) &= \textstyle\sum_{k=1}^{n} 1.s(f_1[k], f_2[k]) \\ dim(f_1, f_2) &= \textstyle\sum_{k=1}^{n} 1.d(f_1[k], f_2[k]) \end{aligned} \tag{4.5}$$

$$s(f_1[k], f_2[k]) = \begin{cases} 1 & f_1[k] = f_2[k] \ \& \ f_1[k] \neq' -' \ \& \ f_2[k] \neq' -' \\ 0 & Otherwise \end{cases}$$

$$d(f_1[k], f_2[k]) = \begin{cases} 1 & f_1[k] \neq f_2[k] \ \& \ f_1[k] \neq' -' \ \& \ f_2[k] \neq' -' \\ 0 & Otherwise \end{cases}$$

where, f_1 and f_2 are two fragments and n is the haplotype length. $s(f_1[k], f_2[k])$ ($d(f_1[k], f_2[k])$) returns '1' if the two variants $f_1[k]$ and $f_2[k]$ are available and identical (different).

The similarity function goes over all variants covered by fragments and checks whether the variants at each polymorphic site are equal in both of the fragments. It computes the number of polymorphic sites in which both fragments carry the same variant; hence, if they carry different variants or one of the fragments carries a missing allele, the polymorphic site is not counted. The dissimilarity function is defined in the same way but it computes the number of different available (non-missing) variants.

Haplotype segment: A haplotype segment is a consensus sequence of a set of fragments ($F' = \{f_1, f_2, f_3, ..., f_t\} \subset F$) which is computed by function **Merge function** and is defined as:

$$s = merge(F') = \underset{s'}{argmax} \sum_{f_k \in F'} sim(f_k, s') \tag{4.6}$$

where F' is a set of fragments and s is the haplotype segment computed as the consensus haplotype of F'. The function which computes s is called merge function. Having computed s, F' is also called **Supporting fragments** of s which is explained later in more detail.

A segment is either a fragment or a consensus sequence of a set of fragments. The former stems from a haplotype and the latter is predicted to be originated from one haplotype. Hence, each segment is supported by one or more fragments, and each of its variants is covered with at least one fragment.

Note that the merge function (*Eq.4.6*) does not solve the Ambiguity of Merging Problem (AoM problem) (see *Section 2.5.1*), while it assumes the input set F' is already not ambiguous and all fragments $f_k \in F'$ stem from one haplotype. The solution for AoM problem is explained in *Section 4.3*.

The aim of haplotype reconstruction is assembling all or a subset of all fragments $F = \{f_1, f_2, f_3, ..., f_m\}$ into assembled haplotypes $H = \{h_1, h_2, h_3, ..., h_P\}$. In an ideal scenario, *i.e.* sufficient accurate signal such as availability of sufficient coverage and long enough and error-free fragments, the aim of haplotype reconstruction is to merge fragments into exactly P haplotypes which are one to one equal to the ground truth ones:

$$sim(h_i, h_j^{True}) = n : h_i \in H,\ h_j^{True} \in H^{True} \tag{4.7}$$

where H and H^{True} are sets of computed and ground truth haplotypes, respectively, and n is the length of the haplotypes.

In reality, due to sequencing errors, mapping errors, and lack of connectivity between alleles, haplotypes may be assembled partially. So, the aim of haplotype assembly is computing a number of haplotypes with maximum length and maximum similarity (or minimum dissimilarity) with the ground truth ones.

4.2.1 Terminologies for *Ranbow* algorithm

We view haplotyping as clustering of fragments in seed regions. One seed region, its SmPs and aligned reads are illustrated in *Figure 4.3* in both nucleotide and coded allele spaces. These seed regions are called 'masks' which is explained below. This helps us to avoid the AoM problem (see *Section 2.5.1*). Our approach looks at the allele combinations in sets of polymorphic sites. To this end, we first define a mask:

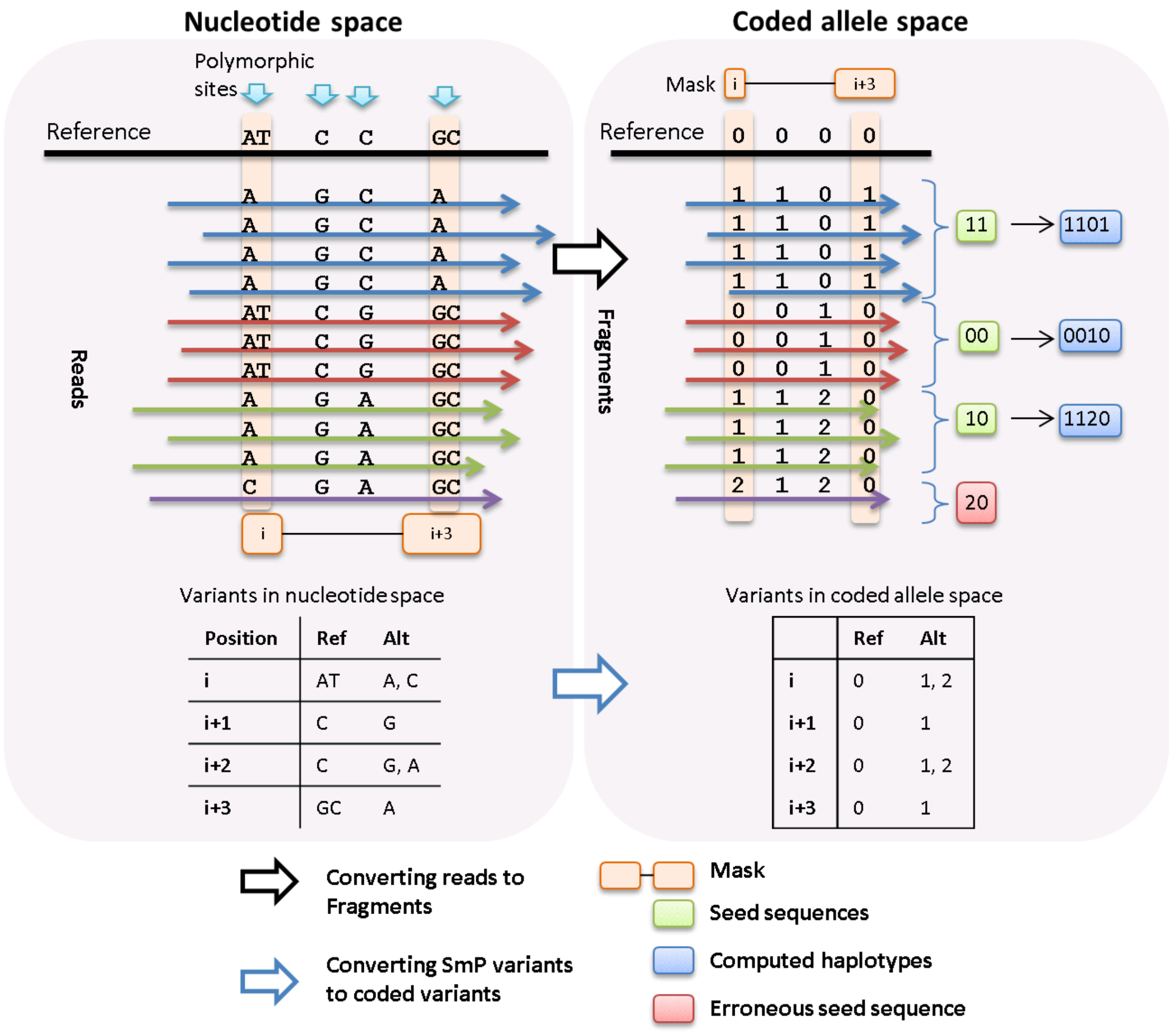
Mask (msk): A mask msk is an ordered set of indices of polymorphic sites. For instance, $(1, 2, 4)$ or $(2, 4, 6)$ are two masks in a haplotype with seven polymorphic sites (all indices are $(1, 2, 3, 4, 5, 6, 7)$). *Figure 4.4-A* shows one mask msk_1 covers 3, 6, and 8 indices; hence, $msk_1 = (3, 6, 8)$.

Seed sequences of a mask (t^{msk}): A seed sequence is a subsequence of a fragment excluding '$-$'s. It indicates the sequence pattern in one fragment at the indices of the mask. For example, having fragments f_1 = -0102-1 and f_2 = 12111-3, $t_1^{(2,4,7)}$ = 001 and $t_2^{(2,4,7)}$ = 213 are two seed sequences of the mask $(2, 4, 7)$. Hence:

$$f_k[msk_i[j]] = a[j] \ : \ a[j] \in t^{msk_i} \tag{4.8}$$

where $a[j]$ is the variant of t^{msk_i} at index of $msk_i[j]$ belonging to f_k. *Figure 4.4-A* shows the corresponding seed sequence of $msk_1 = (3, 6, 8)$ covered by f_1.

Set of seed sequences(T^{msk_i}): T^{msk_i} is defined as all seed sequences covered by a mask msk_i. *Figure 4.4-A* and *Figure 4.4-B* illustrate two examples of regions covered by one and three fragments. f_1 contains '121' at indices $(3, 6, 8)$ so $t_1^{(3,6,8)}$ = 121. In *Figure 4.4-B*, each of f_2, f_3, and f_4 contributes one seed sequence to the mask $msk_2 = (20, 22, 24)$. '001' and '002' are covered by one (f_3) and two (f_2,f_4) fragments, respectively; hence, '001' and '002' have sets of supporting fragment of length one and two. *Figure 4.4-C* indicates that neither of the shown fragments constitute a seed sequence for the mask '34' and '44'.



Variants in nucleotide space

Position	Ref	Alt
i	AT	A, C
i+1	C	G
i+2	C	G, A
i+3	GC	A

Variants in coded allele space

	Ref	Alt
i	0	1, 2
i+1	0	1
i+2	0	1, 2
i+3	0	1

FIGURE 4.3: **From sequence reads to initial haplotypes of a triploid genome in a mask region.** The fragments are clustered according to the seed sequences they carry. There are three clusters; each constructs one haplotype segment (shown in blue boxes). The haplotype segments are supported with four, three, and three fragments, respectively (blue, red, and green fragments). The purple read and fragment are erroneous which is put aside from the assembly. Two tables are shown in the lower part of the figure. The left one is in nucleotide space and the right one is obtained by transforming the left table into coded allele space. Recall that the reference, aligned reads, and list of variants are the inputs.

Depending on the number of reads mapped to a region, one mask may be covered with zero, one, or a few fragments, each of which contributes one seed sequence to a mask. If a read contains variants of all indices indicated in a mask, it contributes one seed sequence to it. Besides, a fragment may contribute several seed sequences to different masks. *Figure 4.5* shows all possible seed sequences contributed by one fragment. In this example, the length of seed sequence varies from two to four. Having four polymorphic sites covered with the fragment $f_k \in F$, the number of seed sequences of lengths two, three, and four are $\binom{4}{2}$, $\binom{4}{3}$, and $\binom{4}{4}$, respectively.

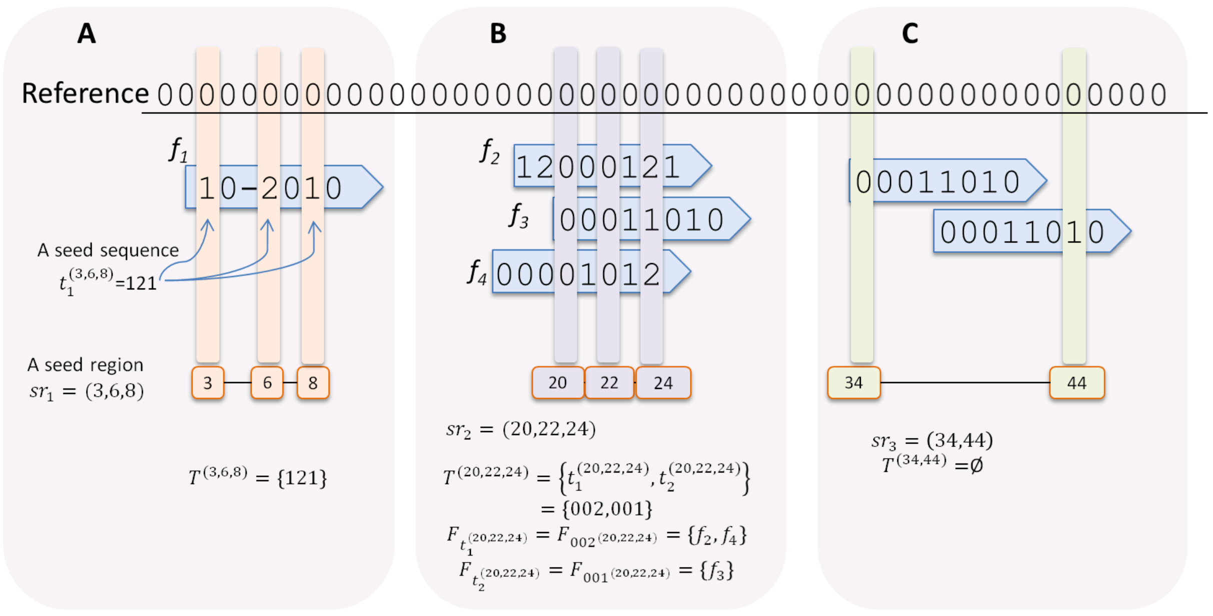

A
B
C
Reference 000000000000000000000000000000000000000000000000
f_1
10-2010
A seed sequence
$t_1^{(3,6,8)}$=121
A seed region
sr_1 = (3,6,8)
3
6
8
$T^{(3,6,8)} = \{121\}$
f_2
12000121
f_3
00011010
f_4
00001012
20
22
24
sr_2 = (20,22,24)
$T^{(20,22,24)} = \{t_1^{(20,22,24)}, t_2^{(20,22,24)}\}$
= {002,001}
$F_{t_1^{(20,22,24)}} = F_{002^{(20,22,24)}} = \{f_2, f_4\}$
$F_{t_2^{(20,22,24)}} = F_{001^{(20,22,24)}} = \{f_3\}$
00011010
00011010
34
44
sr_3 = (34,44)
$T^{(34,44)} = \emptyset$

FIGURE 4.4: **An illustration of masks and their seed sequences.** (A) shows the mask (msk_1), its seed sequence ('121') and the fragment containing this seed sequence (f_1). (B) depicts a mask (msk_2) and the three fragments covering the mask. Since the seed sequence of f_2 and f_4 are identical, both contribute one seed sequence to the mask msk_2, therefore, the set of seed sequences $T^{(20,22,24)}$ contains $t_1^{(20,22,24)}$ and $t_2^{(20,22,24)}$. (C) shows a masks msk_3 with no seed sequence

Supporting fragment of a seed sequence ($F_{t^{msk_i}}$): $F_{t^{msk_i}}$ is a set of fragments which are contributing to a seed sequence t^{msk_i}. therefore:

$$F_{t^{msk_i}} = \{f_k | f_k[msk_i[j]] = a[j]\} \tag{4.9}$$

where t^{msk_i} is a seed sequence, $f_k \in F$, and $a[j]$ is a sequence variant. *Eq.4.9* indicates the variants in seed sequence and fragment at corresponding index are identical. **Set of all masks (MSK^{all}):** MSK^{all} is the collection of all masks with a length of two or more:

$$MSK^{all} = \{msk_i | msk_i \subset \{1, 2, .., n\}, 2 \leq |msk_i|\} \tag{4.10}$$

where n is the number of polymorphic sites. Potentially, there are $\sum_{k=2}^{n} \binom{n}{k} = 2^n - n - 1$ seed sequences available with n polymorphic sites.

4.3 Method

The AoM problem as a critical challenge in polyploid haplotyping was defined and explained in *Section 2.5.1*. To overcome this problem, we need to consider the information

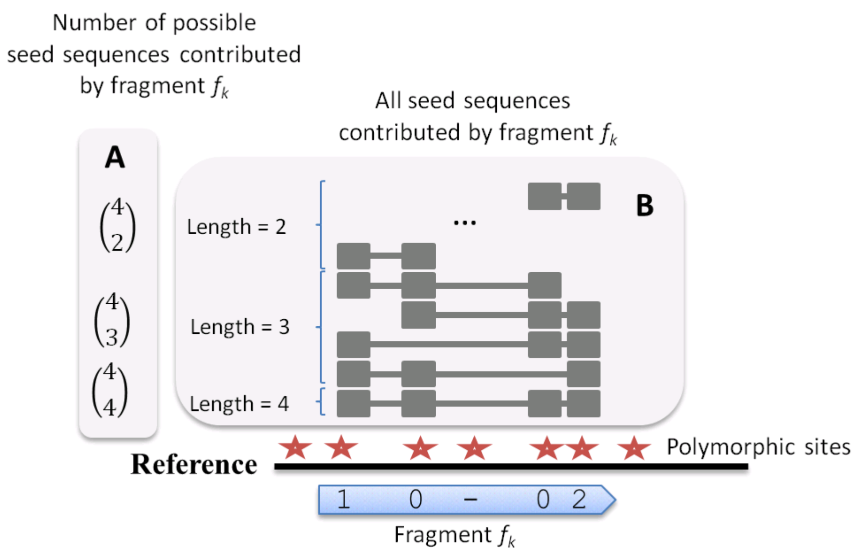


FIGURE 4.5: **All possible seed sequences contributed by one fragment.** The depicted fragment covers four polymorphic sites. The seed sequences of length two, three, and four are shown in panel B. As it is shown in panel A, the number of seeds of length two, three and four are six, three, and one, respectively.

of neighboring reads; hence, we defined masks and their seed sequences in *Section 4.2*. *Figure 4.6* depicts a schematic view of how one mask is phased to the haplotypes according to the supporting fragments of their seed sequences. This figure is illustrated in nucleotide space to make it more understandable. In this example, only polymorphic sites are illustrated (interpolymorphic regions are not shown). The colors in the reference indicate that the genome can be tiled into subregions each of which is similar to one of the haplotypes. One seed sequence represents part of a haplotype, therefore, seed sequences are illustrated in different colors.

A mask and its seed sequences can be considered as a window on the reference sequence and mapped reads. ***Ranbow*** (*Algorithm 5*) finds all masks for which at least one seed sequence is available (*Section 4.3.1* - *Figure 4.6*-*a,b*). Then, it extends the seed sequences with the aid of supporting and neighboring fragments (*Section 4.3.2* - *Figure 4.6*-*c,d*). Adjacent masks may overlap. ***Ranbow*** utilizes a graph data structure with haplotype segments as nodes and overlaps of haplotype segments as its edges. *Section 4.3.3* explains how the overlapping masks can be merged. In order to phase the regions with fewer than P unique haplotypes, *e.g.* two of the chromosome sequences are similar in a region, some restrictions are applied to the mask phasing, which are explained in *Section 4.3.4*.

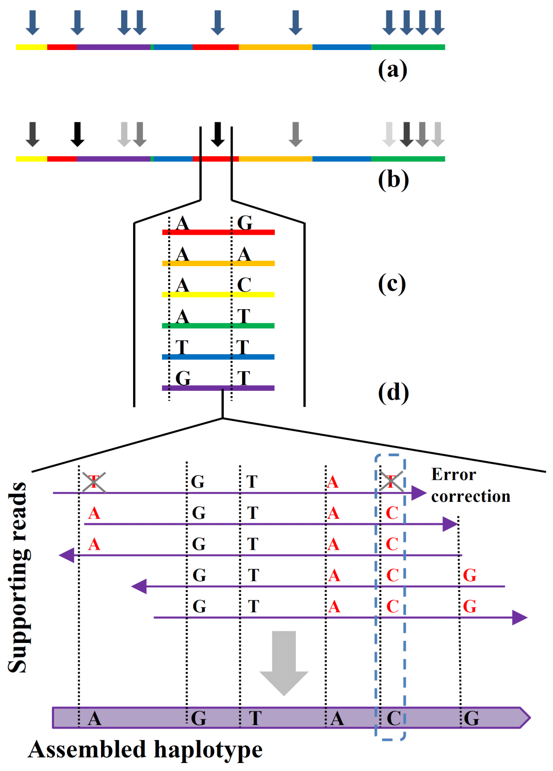


FIGURE 4.6: **Schematic view of a mask phasing for a hexaploid genome.** a) Illustrates a reference genome and possible masks (arrows). The colors in the reference indicate that the genome can be tiled into subregions each of which is similar to one of the haplotypes. b) The arrows are shaded according to the masks' read support. The one with the highest support (shown in black) is selected and phased. c) The mask is phased into its six seed sequences. d) The purple seed sequence and its supporting reads are shown. After error correction the reads are merged into one assembled haplotype.

4.3.1 Mask and seed sequence finding

Each seed sequence is a subsequence of one aligned fragment, and each fragment is sampled from one chromosome; thus, one seed sequence represents one haplotype. P unique seed sequences of a mask represent P unique haplotype segments. For instance, in a hexaploid genome, a mask with six seed sequences is phased into its six segments. The number of seed sequences in a mask could be smaller than P either due to the similarity of alleles in different homologous chromosomes or because of errors from upstream analysis such as base calling, genome assembly, mapping, and variant calling steps. By obtaining all of the masks ($msk_i \in MSK^{all}$s) and their sets of seed sequences (T^{msk_i} for a mask msk_i) with length P or more ($|t^{msk_i}| \geq P$), we phase the genome into **Haplotype Blocks**, shortly **blocks** or b^{msk_i}, of P haplotypes. Considering $|T^{msk_i}|$ as the cardinality of the set T^{msk_i}, or the number of seed sequences in T^{msk_i}, there are three conditions for different values of $|T^{msk_i}|$:

i $|T^{msk_i}| < P$ indicates there is not enough information for phasing the region to P haplotypes. This situation may happen either because some haplotype segments are equal or due to lack of sampled sequence reads, *i.e.* low coverage, or errors in data. The algorithm for dealing with these regions is explained in *Section 4.3.4*.

ii $|T^{msk_i}| = P$ implies that each seed sequence represents one haplotype segment. The mask is phased into its haplotypes.

iii $|T^{msk_i}| > P$ indicates error in the mask. These masks are phased into their seed sequences after applying the error correction step. P seed sequences are kept after error correction, so that the second condition is met. The erroneous seed sequences are detected according to their fragment supports. The smaller the number of supporting fragments, the higher the probability of this being an erroneous seed sequence. In fact, in this case we delete the seed sequences deemed erroneous and keep only P seed sequences. We call this error corrected set $\Theta(T^{msk_i})$ which is defined as:

$$\Theta(T^{msk_i}) = \begin{cases} T^{msk_i}, & |T^{msk_i}| = P \\ T^{msk_i}_{corrected}, & |T^{msk_i}| > P \\ \emptyset, & |T^{msk_i}| < P \end{cases} \tag{4.11}$$

where msk_i is a mask, T^{msk_i} is a set of its seed sequences, and $T^{msk_i}_{corrected}$ is the corrected set of seed sequences of msk_i when $|T^{msk_i}| > P$. When $|T^{msk}| > P$, the seed sequences with less read support are considered as error and are removed to make $|T^{msk}_{corrected}| = P$. *Figure 4.7* illustrates detecting and correcting erroneous fragment for a triploid genome. There are four seed sequences detected for mask $(7, 9)$, so $T^{(7,9)} = \{20, 11, 10, 22\}$ each of which supported with three, two , two , and one fragments, respectively. Hence, $F^{(7,9)} = \{F_{20}, F_{11}, F_{10}, F_{22}\} = \{3, 2, 2, 1\}$. Since we know the genome is triploid as a prior information and F_{22} has the minimum amount of supporting fragment, '22' is considered as error and is removed from $T^{(7,9)}$. Now, $|T^{(7,9)}| = P$ and condition *(ii)* is meet.

Ranbow finds all of the masks and their seed sequences and checks if the conditions *(ii)* or *(iii)* hold. The algorithm proceeds with the following steps: **Mask finding** and **Mask ranking**.

4.3.1.1 Listing masks

Given a fragment list F as input, the goal is obtaining all seed sequences and masks with the following properties:

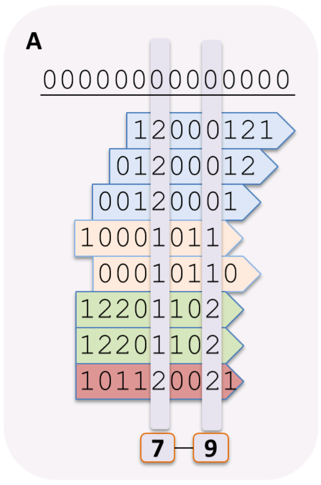


FIGURE 4.7: **An Illustration of errors detection when $T^{msk} > P$ for a triploid genome:** Reads with same seed sequences are depicted in same color. Seed sequence '22' is the erroneous seed sequence because it is supported with one fragment while the other seed sequences are supported with at least two fragments.

i $|msk| \in \{2, 3, ..., k\}$

ii $|T^{msk_i}| \geq P$

Finding all available masks is not trivial. There are potentially $2^n - n - 1$ possible masks, where n is the number of polymorphic sites, but not all of these regions are supported with fragments. To obtain all available masks and their seed sequences, we used a data structure of two nested hash tables. The keys in the outer hash table, HR (Hash table of masks), are masks and the values are hash tables. The inner hash table keeps seed sequences as a key and a list of supporting fragment as the value. For instance, $|HR_{00}^{(0,2)}| = 7$ means there are 7 fragments supporting seed sequence '00' in loci 0 and 2.

To fill the mentioned data structure, ***Ranbow*** goes through the fragments list. Each fragment contributes a number of seed sequences to their corresponding masks. A fragment with k non-missing alleles contributes $2^k - k - 1$ seed sequences to different masks. For example, the fragment f_i = 00-1 contributes 1 support to each of the following seed sequences and masks:

$$\begin{aligned} t^{(0,1)} &= 00 \\ t^{(0,3)} &= 01 \\ t^{(1,3)} &= 01 \\ t^{(0,1,3)} &= 001 \end{aligned}$$

Consequently it contributes one support to:

$$HR_{00}^{(0,1)}$$
$$HR_{01}^{(0,3)}$$
$$HR_{01}^{(1,3)}$$
$$HR_{001}^{(0,1,3)}$$

Using this data structure, ***Ranbow*** fills HR independent of the position of f_is, their lengths, and the scaffold size.

This data structure provides a fast way to find all of the masks, seed sequences, and the supporting fragments. This algorithm provides all available masks (we call this set MSK) and ignores masks with no supporting fragments.

Two masks, msk_i and msk_j, may overlap at one or more loci. By phasing msk_i, there are two possibilities for phasing msk_j.

i $msk_j \subset msk_i$: msk_j is already phased by msk_i; thus, there is no need to phase the region again.

ii $msk_j \not\subset msk_i$: msk_j needs to be phased.

Condition i indicates that the order of phasing masks is important and influences the results. This raises the question of which mask has higher priority for being phased.

4.3.1.2 Mask ranking

Given MSK, T_{msk}, and $F_{t^{msk}}$, ***Ranbow*** ranks the masks according to the least supported seed sequence of the mask. The fitness for each msk_i is defined as:

$$\psi(msk_i) = \min_{t^{msk_i} \in \Theta(T^{msk_i})} |F_{t^{msk_i}}| \tag{4.12}$$

where Θ is defined in *Eq.4.11* as the corrected set for T^{msk_i}, $F_{t^{msk_i}}$ is a set of supporting fragments for seed sequence t^{msk_i}.

This defines a fitness function $\psi(msk_i)$, which computes the fitness according to the read support for the P^{th} highest supported seed sequence. Each seed sequence has a number of supporting fragments. We sort them descendingly and the P^{th} number in the sorted list is considered as the fitness of the mask. The reason behind this is that the masks with more support for the weakest seed sequence are more reliable to be phased first. For instance, for the situation depicted in *Figure 4.7*, $msk_i = (7, 9)$ and $\psi((7, 9)) = 2$ due to the fact that the least supported seed sequences are supported with two fragments, which are colored in green and yellow. The red fragment is an error because the genome is assumed to be triploid.

An error in a supporting fragment may occur either in masks or non-masks positions. For instance, consider f_i = 12111-3 as a fragment, $msk = (1, 2, 4)$ as one mask, and $t^{(1,2,4)} = 121$ as one of its seed sequences. An error may occur in positions 1, 2, or 4, *i.e.* error in masks, or in positions 3, 5, or 7, *i.e.* error in non-mask. Note that, the error correction function Θ only corrects the errors in masks positions. The variants in non-masks indices can be corrected via merge function which was defined in *Eq.4.6* (see details in *Section 4.3.2)*.

The masks are ranked based on the fitness defined in *Eq.4.12*. For phasing a mask we prefer to phase the one with higher score. The higher the score is, the higher the priority for a masks to be phased. We use MSK^{ranked} notation when the mask is sorted as explained.

4.3.2 Phasing a mask and its extension

Constructing a list of ranked masks, here, we explain how a mask is phased, and how the segments in a phased mask are elongated.

4.3.2.1 Phasing a mask with P seed sequences

Given a mask msk_i and its P seed sequences ($t_j^{msk_i} \in T^{msk_i}$) each of which are supported with supporting fragments $F_{t_j^{msk_i}}$ as input, ***Ranbow*** phases msk_i into its P haplotype segments (defined in *Section 4.2)* and collects them in a set called a **Haplotype Block** b^{msk_i}:

$$b^{msk_i} = \{s_j^{msk_i} | s_j^{msk_i} = merge(F_{t_j^{msk_i}})\} \tag{4.13}$$

where $s_j^{msk_i}$ is a haplotype segment and merge function is defined in *Eq.4.6*.

We define a haplotype block as:

Haplotype Block (b^{msk_i}): A set of haplotype segments ($s_j^{msk_i}$) with the following properties: 1) Initiated from one mask (msk_i). 2) The haplotype segments are constructed from the supporting fragments ($F_{t_j^{msk_i}}$) of the seed sequences of the mask 3) The haplotype segments are elongated by neighboring fragments.

Here we discuss the elongation of a haplotype block by overlapping fragments. Recall that, a matching overlap does not provide enough information to apply the merge function due to the AoM problem (*Section 2.5.1*). Here, this problem is addressed by recruiting the information of the other overlapping sequences.

For this purpose, we define the following criteria for a segment s_i and a fragment f_k, which is called a **matching overlap**:

$$\begin{aligned} &\text{i) } sim(s_i, f_k) > 0 \\ &\text{ii) } dim(s_i, f_k) = 0 \end{aligned} \tag{4.14}$$

where *sim* and *dim* functions are defined in *Eq.4.5*. Recall that a segment could be a fragment or a set of fragments (*Section 4.2-haplotype segment*); hence, the matching overlap can be checked between one fragment and one segment or two segments. Let s_i and s_j, be two different segments belonging to b^{msk_i}, and f_k be an overlapping fragment, we are looking for the criteria in which f_k and s_i can be merged with no ambiguity. First let's discuss if f_k shares a matching overlap with s_i, then, the following cases may happen:

I f_k shares non-matching overlaps with all of s_j's; hence, there is one possibility for f_k to be merged with, and that is s_i. The f_k has to have non-matching overlap with $P-1$ segments and have matching overlap with s_i in order to be merged with no ambiguity. This assures that f_k belongs to one and only one haplotype. We call this kind of overlap a **unique match** (see *Figure 4.8* for an illustration of unique match in nucleotide space) between s_i and f_k.

II f_k shares a matching overlap with s_j; so there is no signal to explain whether f_k belongs to s_i or belongs to s_j.

III The same applies if f_k does not share an overlap with one segment s_j. This means there is no signal to distinguish if the overlap between s_j and f_k is matching or not; therefore merging would be ambiguous.

Secondly, f_k may have P mismatching overlap, which provides the same condition for a unique match by finding the errors in one of the overlaps. We define the selecting scoring function, Φ in order to distinguish which overlap is more likely to be the one f_k could be merged with:

$$\begin{aligned} s_{selected} &= \underset{s_i \in b^{msk}}{argmin}\ \Phi(s_i, f_k) \\ \Phi(s_i, f_k) &= sim(s_i, f_k) - dim(s_i, f_k)^2 \end{aligned} \tag{4.15}$$

where $s_{selected}$ is the selected segment to be merged with f_k, $\Phi(s_i, f_k)$ is the scoring function evaluating which segment s_i is more likely to have originated from the haplotype which f_k belongs to, therefore, can be merged with f_k. Φ is designed to penalize the erroneous haplotypes and to select the segment with higher similarity and lower dissimilarity with f_k. This condition is also called a unique match, in the following steps of the algorithm.

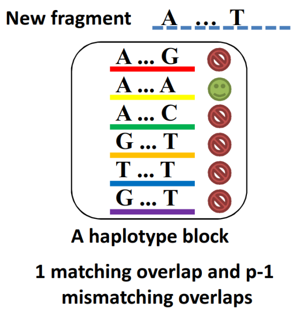


FIGURE 4.8: **Uniquely matched overlap between a fragment and a set of haplotypes.** The only matching overlap between the fragment and the block is "A". Since there is just one matching overlap and five mismatching overlaps, the new fragment can be merged with the yellow haplotype segment.

4.3.2.2 Block extension

Given haplotype segments of a block $s_i \in b^{msk_t}$ and a fragment list F, s_i's can be extended using neighboring fragments. We define a set of overlapping fragments of a block (F') as follows:

$$F' = \{f'_j | f'_j \in F, \forall s_i \in b^{msk_t} : sim(f'_j, s_i) + dim(f'_j, s_i) > 0\} \tag{4.16}$$

where similarity and dissimilarity functions were defined in *Eq.4.5*. For block extension, ***Ranbow*** selects a fragment f'_j and a segment s_i such that:

$$f'_j, s_i = \underset{f'_j \in F', s_i \in b^{msk_t}}{argmax} \Phi(f'_j, s_i) \tag{4.17}$$

where $\Phi(f'_j, s_i)$ is defined in *Eq.4.15*. *Eq.4.17* indicates ***Ranbow*** selects f'_j and s_i such that their overlap provides the maximum score. Then the supporting fragment of s_i, *i.e.* F^{s_i}, is updated, and consequently, updated F^{s_i} results in a new s_i:

$$\begin{aligned} F^{s_i} &\leftarrow F^{s_i} \cup \{f'_j\} \\ s_i &= merge(F^{s_i}) \end{aligned} \tag{4.18}$$

Now, the haplotype block is extended in length so it may overlap with new fragments. These steps are done iteratively until the segments in a haplotype block converge.

In other words, at each iteration ***Ranbow*** selects a fragment, if possible, to merge it with one haplotype. After merging, the new fragment overlaps are introduced and the scores between the segments in the block and the overlapping fragments are updated. This leads to the block extension as it is shown in *Figure 4.9*.

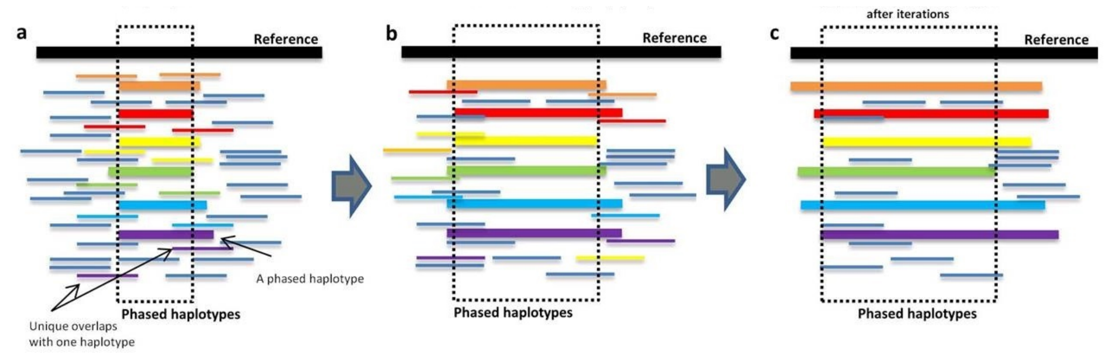


FIGURE 4.9: **The seed extension process:** a) A constructed seed and its corresponding haplotypes. b) Seed extension by uniquely matched reads c) iteratively the blocks are getting longer and the final block is constructed.

4.3.3 Merging overlapping and connected masks

Block extension is applied on all phased masks. Although the two masks msk_i and msk_j do not share any overlap, their extended blocks namely, b^{msk_i} and b^{msk_j}, might overlap. There might be fragments (from single-end, paired-end, and mate-pair reads) which are partly used in $s_i \in b^{msk_i}$ and partly used in $s_j \in b^{msk_j}$. In other words, one fragment f_i could be a member of several segments of different blocks ($f_i \in F^{s_j} \cap F^{s_k} | j \neq k$); hence, it provides information to merge the haplotype segments it belongs to. In the next step, we recruit these connections to merge the overlapping segments to elongate the segments. We call this procedure **Merging overlapping haplotype segments**.

4.3.3.1 Overlapping haplotype segments

Given a list of blocks $B = \{b^{msk_1}, b^{msk_2}, ..., b^{msk_b}\}$ and the supporting fragments of the segments in these blocks, the aim is using the connections between the segments for elongation of haplotypes. We model this problem as a weighted k-partite graph $G(V, E)$, in which:

$$\begin{aligned} V &= \{s_x | s_x \in b^{msk_t}, 1 \leq t \leq b\} \\ E &= \{e(s_x, s_y) | s_x \in b^{msk_i}, s_y \in b^{msk_j}, i \neq j\} \end{aligned} \tag{4.19}$$

where s_x and s_y are two haplotype segments belonging to b^{msk_i} and b^{msk_j}, respectively, as the nodes of the graph, $e(s_x, s_y)$ is an edge between s_x and s_y. We define weights, w's, for the edges which are calculated as the number of fragments which belong to the intersection of supporting fragments: $F^{s_x} \cap F^{s_y}$. We draw an edge between two nodes if $F^{s_x} \cap F^{s_y} \neq \emptyset$.

Graph G is multipartite, each b^{msk_t} is a partition, so there is no edge connecting the nodes inside a partition. Nodes are partitioned according to the haplotype blocks. In the ideal scenario, *i.e.* no error in input data, each path or cycle in this graph indicates an elongated haplotype segment, called here **desired paths** and **desired cycles**. There might be paths occurring due to errors in the graph, *i.e.* **conflicting paths**. *Figure 4.10* depicts a schematic view of graph G.

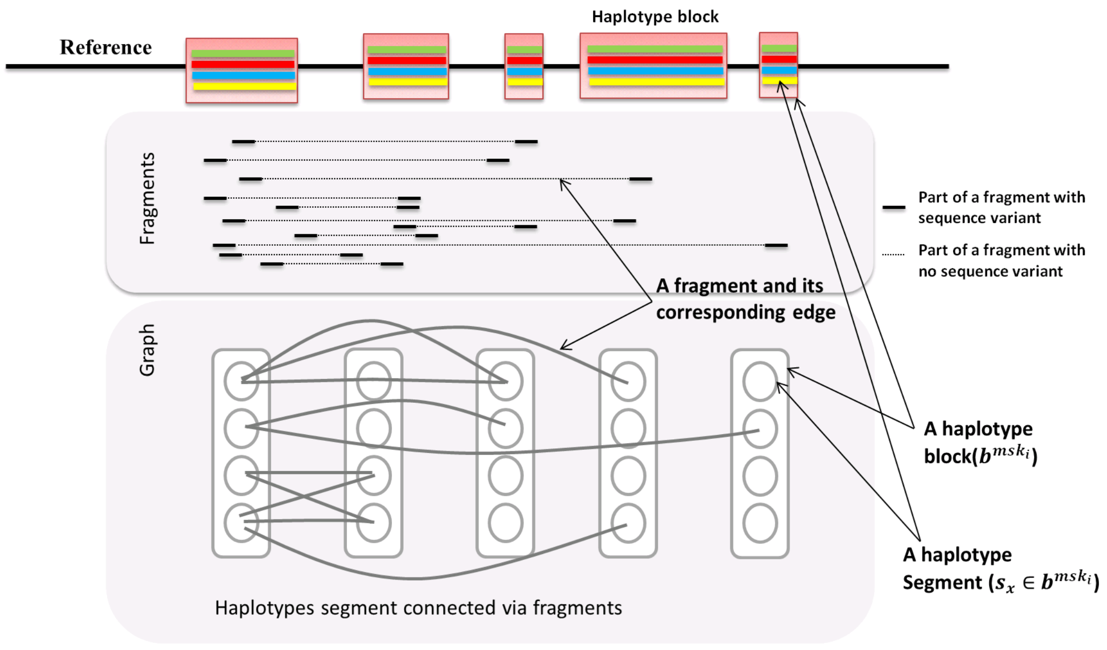

Haplotype block
Reference
Fragments
Part of a fragment with sequence variant
Part of a fragment with no sequence variant
A fragment and its corresponding edge
Graph
A haplotype block(b^{msk_i})
A haplotype Segment ($s_x \in b^{msk_i}$)
Haplotypes segment connected via fragments

FIGURE 4.10: **Schematic view of graph G:** This figure shows a number of fragments and their corresponding edges in graph G. The fragments with missing alleles may be obtained from paired reads. If two reads of a pair are mapped into two haplotype segments of different blocks, one edge is assigned between the nodes.

4.3.3.2 Conflicting paths, desired paths, and desired cycles

Conflicting paths: A conflicting path is defined as a path which connects two nodes of the same partition. This means two segments of one block should be merged which violates the definition of segments in haplotype blocks and also the definition of partitions in a k-partite graph. For instance, *Figure 4.11-B* depicts three conflicting paths. $s_0^{msk_1} \rightarrow s_0^{msk_2} \rightarrow s_1^{msk_1}$ is a conflicting path because this path indicates $s_0^{msk_1}$ and

$s_1^{msk_1}$, which are two distinct segments, are essentially originated from one haplotype. This creates a conflict. Conflicting paths occur due to errors in the set of edges (E).

Desired paths: A path which is not conflicting is called a desired path.

Desired cycles: Edges of a cycle are supporting each other; this gives us a better clue for the connectivity of haplotypes. For instance, for a cycle with length three of $s_0^{msk_1} \rightarrow s_1^{msk_2} \rightarrow s_0^{msk_3} \rightarrow s_0^{msk_1}$ (shown in *Figure 4.11-A(Blue)*) every path between a number of nodes, *e.g.* $s_0^{msk_1} \rightarrow s_1^{msk_2} \rightarrow s_0^{msk_3}$, is supported by another path, *e.g.*, $s_0^{msk_3} \rightarrow s_0^{msk_1}$; thus, firstly, we base the elongation step on these cycles.

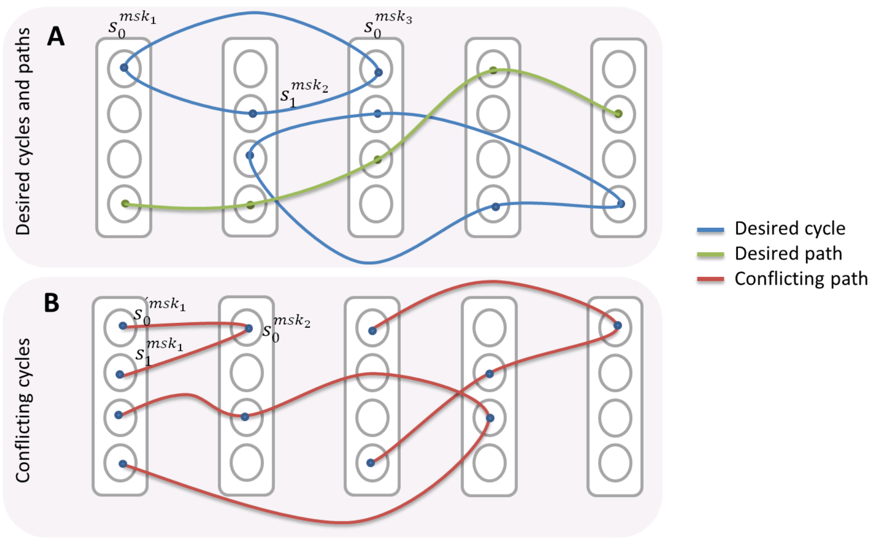


FIGURE 4.11: **Desired cycles, desired paths, and conflicting paths.** A and B illustrate two desired cycles (blue), one desired path (green), and three conflicting paths (red). The conflicting paths are those containing two nodes of the same partition. In desired cycles the connectivity between two nodes are not only supported with a direct edge but also supported through the path of other nodes of the cycle.

4.3.3.3 Merging the haplotype segments in non-conflicting cycles

In order to find desired cycles and paths, first the graph should be constructed which is a costly task due to the following reasons:

i Finding all non-conflicting cycles in the graph is of high time complexity

ii It is costly to find all edges in G because all pairs of segments need to be checked to see if they share fragments or not.

Moreover, the probability of having an edge between distant haplotype segments is low. Therefore, we designed a greedy algorithm which avoids constructing the whole graph. It, first, constructs the graph partially and finds cycles with the length of three, called here **triangles**. For this purpose, ***Ranbow*** sorts haplotype blocks ($B = \{b^{msk_1}, b^{msk_2}, ..., b^{msk_b}\}$) based on their starting positions and constructs a new list of blocks called B^{sorted}. Recall that a haplotype block which is defined in *Section 4.3.1*, is a collection of haplotype segments which are obtained from a mask. The starting position of a block is the smallest starting position of its segments. Let l be the length of a sliding window covering a number of neighboring blocks in B^{sorted}. In *Figure 4.13*, red rectangles are the sliding windows with the size of four, $l = 4$. Then, ***Ranbow*** searches for the triangles and desired paths at each sliding window. Considering a sliding window covers l blocks where b^{msk_i} is the first block and b^{msk_j}'s are the other blocks in the same sliding window, ***Ranbow*** takes b^{msk_i} and then, finds all of the edges starting from $s_x \in b^{msk_i}$ and connecting to $s_y \in b^{msk_j}$. Let E' be a set of edges which connect s_x to s_y (*Figure 4.13-solid gray edges*):

$$\begin{aligned} E' = \{e(s_x, s_y, w_{(s_x,s_y)}) | w_{(s_x,s_y)} = |F^{s_x} \cap F^{s_y}|, w_{(s_x,s_y)} > 0\} \\ \text{where } \forall s_x \in b^{msk_i}, s_y \in b^{msk_j} \text{ with } (s_x, s_y, w) \in E \Rightarrow i \neq j \end{aligned} \tag{4.20}$$

where $w_{(s_x,s_y)}$ is the number of fragments shared between s_x and s_y ($|F^{s_x} \cap F^{s_y}|$). ***Ranbow*** searches for a pair of edges such that they start from one node, s_x, and end in two nodes, s_y and s_z where s_y and s_z are not in the same block. *Figure 4.12* depicts this situation. Then, ***Ranbow*** checks whether there is an edge between s_y and s_z or not. If this edge exists, s_x, s_y and s_z construct a triangle. ***Ranbow*** seeks to find a triangle with **maximum fragment support** as:

$$\begin{gathered} s_x, s_y, s_z = argmax\ w(s_x, s_y) + w(s_x, s_z) + w(s_y, s_z) \\ \text{where } s_x, s_y, s_z \in V, \\ \text{and } s_x \in b^{msk_i}, s_y \in b^{msk_j}, s_z \in b^{msk_k} \end{gathered} \tag{4.21}$$

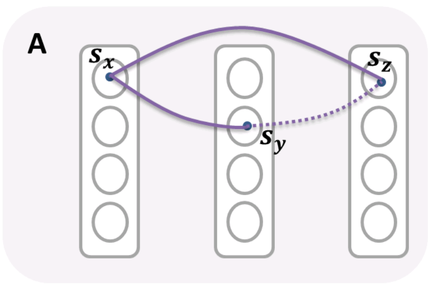


FIGURE 4.12: **Searching for triangles:** This figure depicts a triangle starting at s_x and ends at s_y and s_z. ***Ranbow*** checks if there is an edge between s_y and s_z (dotted line) to form a triangle.

Having found a triangle with maximum support, ***Ranbow*** merges s_x, s_y, and s_z (*Figure 4.13-solid blue lines*) and constructs a new haplotype segment and replaces the s_x with the updated haplotype (s_x^{new}) such that:

$$\begin{aligned} F_{new}^{s_x} &= F^{s_x} \cap F^{s_y} \cap F^{s_z} \\ s_x^{new} &= merge(F_{new}^{s_x}) \\ s_x &\leftarrow s_x^{new} \end{aligned} \tag{4.22}$$

where F^{s_x}, F^{s_y}, and F^{s_z} are the sets of supporting fragments for s_x, s_y, and s_z, respectively. This step results in an updated b^{msk_i} and the removal of s_y and s_z; however, ***Ranbow*** keeps track of which nodes are merged and which are removed. In the example above, ***Ranbow*** keeps the information that s_x, s_y, and s_z are merged into s_x (s_y, and s_z are removed). Moreover, it keeps the information that s_y and s_z belonged to b^{msk_j} and b^{msk_k}, respectively. This information is useful for the next iterations in order to identify conflicting cycles when s_x is be a candidate for merging to another segment. After this step, the algorithm goes to the next iteration and updates E' and applies the same steps until no triangle can be found in E'. As the elongation is done for b^{msk_i}, then the same procedure is applied for $b^{msk_{i+1}}$ with a window shifted one block to the right. The algorithm stops when the sliding window arrives and processes the last group of blocks and all of the triangles are considered. *Figure 4.13* represents a simple example with eight haplotype blocks and explains how the triangles are gradually formed in a sliding window of length four.

4.3.3.4 Merging through direct connections between haplotype segments

The next step is elongation through direct connections. ***Ranbow*** sorts all edges based on their weight descendingly and iterates over the sorted list of edges. At each iteration, ***Ranbow*** selects one edge and checks if merging two nodes of the edge results in a conflict or not. If there is no conflict then the nodes are merged and the graph gets updated. Therefore, having $e(s_x, s_y, w_{(s_x,s_y)}) \in E$ with maximum $w_{(s_x,s_y)}$, ***Ranbow*** merges s_x and s_y nodes such that:

$$\begin{aligned} F_{new}^{s_x} &= F^{s_x} \cap F^{s_y} \\ s_x^{new} &= merge(F_{new}^{s_x}) \\ s_x &\leftarrow s_x^{new} \end{aligned} \tag{4.23}$$

and s_y is removed from V.

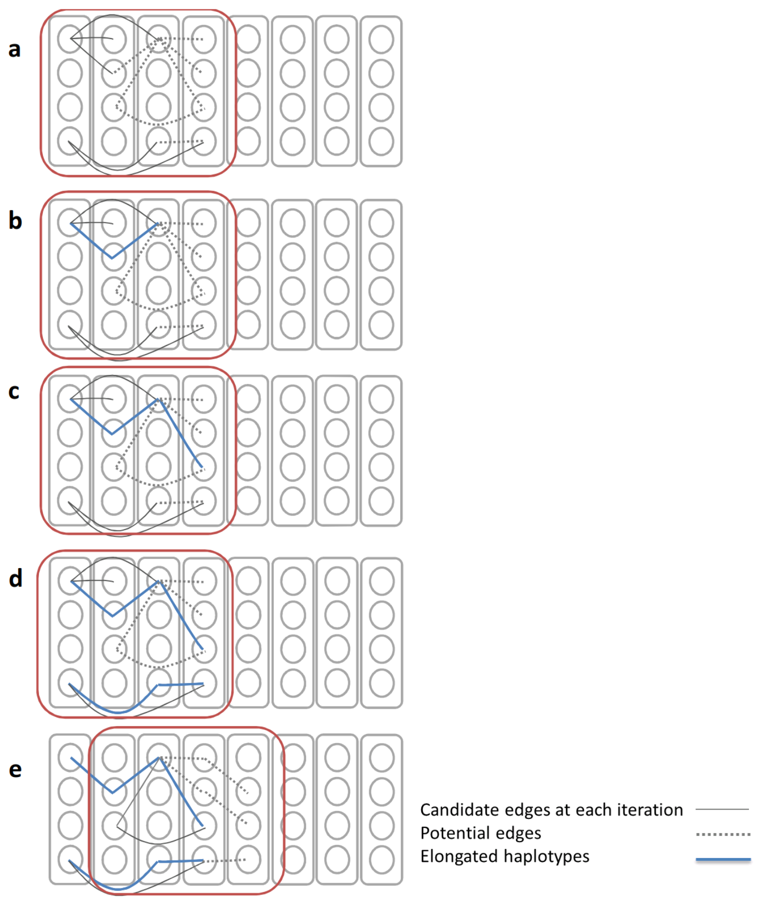


FIGURE 4.13: **Connecting haplotype segments of different blocks in an iterative manner.** Edges are the connections obtained via reads. Red rectangles are sliding windows with length of four. Solid gray lines are the edges beginning from the leftmost block of a sliding window. The dotted lines are edges, and the blue edges are the newly formed haplotypes. The candidate edges at each step are checked to find if they construct a triangle or not. The most supported triangle at each step is converted to one haplotype segment.

Note that, in order to find conflicts, the algorithm checks if a segment consists of many segments, which were merged in former iterations, or it consists of one segment. To make it more clear, let's follow the example below which is illustrated in *Figure 4.14*. Assume $s_x = merge(s_x, s_y)$ and $s_p = merge(s_p, s_q)$ which were done in previous iterations, and assume s_x, s_y, s_p, and s_q belonged to b^{msk_i}, b^{msk_j}, b^{msk_j}, and b^{msk_k}, respectively. Now consider s_x and s_p are the candidates for being merged at current iteration. ***Ranbow*** identifies the origin of s_x and s_p to check that there will not be any conflict caused by merging these two segments. Since s_y and s_p belong to one block, *i.e.* b^{msk_j}, ***Ranbow*** does not merge s_x and s_p and considers this condition as a conflict.

Now assume that s_x and s_y can be merged with no conflict. To update E and to delete s_y, we only use s_y and its connecting nodes (such as s_z) considering the following conditions:

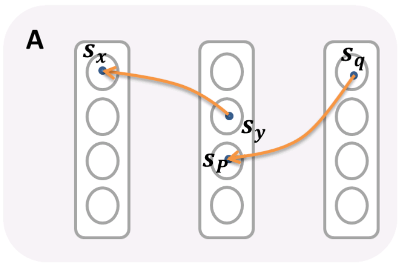


FIGURE 4.14: **Conflicts in merging two segments** This figure shows how a conflict occurs by merging s_x and s_p if s_x consists of s_x and s_y, and also s_p consists of s_p and s_q. Merging s_x and s_p results in merging s_y and s_p which causes a conflict of merging two segments of a block

$$\begin{aligned}
&\text{i) if } e(s_y, s_z, w_{(s_y,s_z)}) \in E \ \ , \ \ e(s_x, s_z, w_{(s_x,s_z)}) \notin E \implies E \leftarrow E + e(s_x, s_z, w_{(s_y,s_z)}) \\
&\text{ii) if } e(s_y, s_z, w_{(s_y,s_z)}), e(s_x, s_z, w_{(s_x,s_z)}) \in E \implies w_{(s_x,s_z)} \leftarrow w_{(s_x,s_z)} + w_{(s_y,s_z)}
\end{aligned} \tag{4.24}$$

where $e(s_y, s_z, w_{(s_y,s_z)})$ is an edge between s_y and s_z with the weight of $w_{(s_y,s_z)}$ and E is the set of all edges. *Eq.4.24* explains how ***Ranbow*** acts if there is an edge between s_y and another node, called s_z, by merging s_y to s_x (explained in *Eq.4.23)*.

Eq.4.24-i indicates that there is no edge between s_x and s_z, therefore by merging s_y to s_x, the edges between s_y and other segments are added to the s_x. *Eq.4.24-ii* indicates that there is an edge between s_x and s_z, so by merging s_y to s_x, just $w_{(s_x,s_z)}$ has to be updated.

The above steps are done iteratively on the list of sorted edges. At each step $e(s_i, s_j, w)$ with maximum weight is selected and its nodes are merged. By merging two nodes and their corresponding edges, E is updated. The algorithm stops when $E = \emptyset$ or the weight of edges at an iteration goes lower than a predefined threshold.

4.3.4 Phasing the regions with fewer than P haplotypes

All regions with P distinct sequences have been phased to P haplotypes so far. In a P-ploid genome there is the possibility of having regions with fewer than P haplotypes. This may happen if the heterozygosity of a region is low such that the reads cannot extend enough to find exactly P unique sequences. Therefore, these regions can be phased to $p < P$ haplotypes. We use masks again to find haplotype blocks with $p < P$ seed sequences. Having MSK^{all} as the collection of masks with at least one fragment support for each seed sequence, we define masks with p seed sequences each as:

$$\begin{gathered} MSK_p = \{msk | msk \in MSK^{all}, \forall msk_i : |T^{msk_i}| = p, 2 \leq p \leq P\} \\ \text{hence} \\ MSK^{all} = MSK_P + MSK_{P-1} + MSK_{P-2} + ... + MSK_2 \end{gathered} \tag{4.25}$$

where T^{msk_i} is the set of seed sequences of the mask msk_i. If one region cannot be phased into P haplotypes, firstly, we check if it can be phased into $p = P - 1$ haplotypes. For this purpose ***Ranbow*** uses MSK_{P-1}. The algorithm is designed such that it starts from $p = P - 1$ and goes down to $p = 2$. For each p, it iterates over sorted $msk_i \in MSK_p$, and checks if msk_i is already phased or not. If the mask is not phased so far, ***Ranbow*** phases it into p haplotypes.

The extension part cannot be applied here; the reason is explained in the simplest scenario of $p = P - 1$ and $p = P - 2$. Let's assume:

$$b^{msk_i} = \{b_1^{msk_i}, b_2^{msk_i}, ..., b_p^{msk_i}\} \ : \ msk_i \in MSK_{P-1} \tag{4.26}$$

as a haplotype block with p segments. The condition shown in *Eq.4.26* indicates that one of the segments, for instance, $b_k^{msk_i}$ has two copies. Let's assume that $b_k^{msk_i}$ can be found from the coverage of segments. *Figure 4.15* depicts this condition in a tetraploid genome with two blocks of three segments each. Let's assume that the weights of edges are known and shown in the *Figure 4.15-A*. There are two possible scenarios depicted in *Figure 4.15-A*. There is no sufficient evidence to infer which of these scenarios is the truth. This introduces an ambiguity for merging haplotype segments. The situation becomes more complicated if $p < P - 1$. For instance, if $p = P - 2$, firstly we need to decide if there are two copies of two haplotype segments or three copies of one of them (*Figure 4.15-B*). Then, in either scenario, the ambiguity explained above prevents extension of haplotypes. Therefore, we leave the phased haplotypes with no extension in this step.

4.3.5 Summary

In this chapter, we explained a novel method, called ***Ranbow***, for reconstructing haplotypes of a polyploid genome from sequence reads. Firstly, we introduced the mathematical problem definition of polyploid haplotype assembly. Then, ***Ranbow*** is explained in detail. In the next chapter, we evaluate ***Ranbow*** based on various real and simulated datasets and compare it with available state-of-the-art methods.

Ranbow algorithm

```
Input : Fragment list $F$
Output: Assembled haplotype $H$
$MSK^{all}, T_{msk}, F_{t^{msk}} \leftarrow$ Mask finding($F$)
$MSK_P^{ranked} + MSK_{P-1}^{ranked} + MSK_{P-2}^{ranked} + ... + MSK_2^{ranked} \leftarrow$ Mask ranking($MSK^{all}$)
$B = \emptyset$
for $msk_i \in MSK_P^{ranked}$ do
    if $msk_i$ not already phased then
        $b^{msk_i}$=Phase a mask($msk_i, T^{msk_i}$)
        $b_{extended}^{msk_i}$= Block extension($b^{msk_i}, F$)
        $B = B + b^{msk_i}$
    else
        Ignore seed $msk_i$
    end
end
$G(V, E) \leftarrow$ Convert to graph ($B$)
for sliding window $w \in$ all sliding windows do
    for block $b^{msk_i} \in w$ do
        for triangles $c$ starting $from$ $b^{msk_i}$ do
            if $c$ is non-conflicting cycle then
                $B \leftarrow$Merging cycles($c, B$)
            else
                Ignore seed $c$
            end
        end
    end
end
for $e \in sorted(E)$ do
    $b \leftarrow$Merge haplotype segments($e, B$)
end
for $p = P - 1$ downto 2 do
    for $msk_i \in MSK_p^{ranked}$ do
        if $msk_i$ not already phased then
            $b^{msk_i}$=Phase a mask($msk_i, T^{msk_i}$)
            $B = B + b^{msk_i}$
        else
            Ignore seed $msk_i$
        end
    end
end
```

Algorithm 5: *Ranbow* algorithm

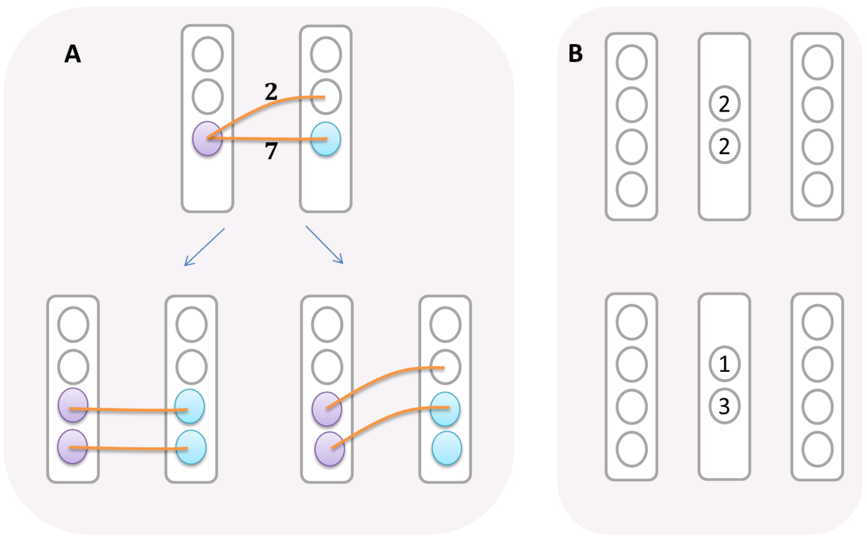


FIGURE 4.15: **Different possibilities for phasing the regions with fewer than P sequence patterns.** In this example $P = 4$. (A) depicts two regions each of which phased into three haplotypes. The colored nodes indicate the nodes with two copies. The weights of edges show the number of fragments connecting the nodes. The two possibilities of connecting nodes are shown in (A). The left one assumes that the connection with the weight two is an error. The right one indicates there is no error and the weights are distributed unevenly. (B) illustrates two possible phases of a region with two unique segments, and it is not known which one is true.

Chapter 5

Evaluation and results

5.1 Introduction

In the Results Section, the performance of ***HapCompass***[47], ***SDhaP***[48], ***H-PoP***[49], and ***Ranbow*** (*Chapter 4*) are assessed on real and simulated data. As a real dataset, we used hexaploid sweet potato sequencing data. Long Roche 454 reads are recruited in order to evaluate haplotype assembly of Illumina short reads. We also generated two simulated datasets, with various read insert sizes, inspired by sweet potato and tetraploid CBU genomes. The performances of all methods are evaluated on mentioned datasets concerning accuracy, length of assembled haplotypes and execution time.

5.2 Results

Ranbow is implemented using Python 2.6.8. and is available online[1]. It takes Fasta (collection of reference sequences), VCF (collection of sequence variants), and BAM (collection of aligned reads) files as input and returns the list of aligned haplotypes in BAM and *hap* format (*hap* format file keeps the haplotypes in coded allele space; please see *Section 4.2* for the formal definition of coded allele space). Since we aimed to develop a method compatible with high-throughput sequence reads needed for *de novo* assembly, distributing data to multiple cores is also automated. ***Ranbow*** accepts the number of available cores as input parameter, and groups the scaffolds such that it tries to minimize the estimation of maximum running time for the busiest core. The assignment of scaffolds into cores is not a trivial problem, so we designed and implemented a heuristic method for this purpose.

[1] http://www.molgen.mpg.de/ranbow

To evaluate ***Ranbow***, we compared it to the state-of-the-art methods ***HapCompass***, ***SDhaP***, and ***H-PoP*** regarding accuracy, length of the assembled haplotype, and runtime. We used one real and two simulated datasets, each of which contains 50 scaffolds with different lengths. The evaluation contains three parts. We firstly compare all methods on a smaller dataset due to the very high execution time of some methods on large datasets. Then ***H-PoP***, which performs better than the other available approaches, is comprehensively compared to ***Ranbow***. Finally, the performance of ***Ranbow*** on the real sweet potato dataset is compared to the corresponding simulated one.

In order to convert input data from the nucleotide space to the coded allele space (explained in *Section 2.2*) for the sake of decreasing time and space complexity, ***Ranbow*** extracts a list of fragments from the input sequence reads and variants (in BAM and VCF files). Based on the sequencing throughput and heterozygosity of the genome, the volume of the read-file could be reduced hundredfold, *e.g.*, in sweet potato genome, it is reduced from 657Gb of mapped reads down to 1.6Gb of aligned fragments. The VCF and the fragment files are indexed to speed up the search process to balance scaffold distribution on a processor farm. The scaffolds are assigned optimally to the processors according to their length to minimize the maximum run time among all cores. ***Ranbow*** generates *hap* file format, which contains the assembled haplotype segments (at coded allele space) and their properties. The haplotypes at nucleotide space are generated afterwards and converted to sorted and indexed BAM format, which can be uploaded to the genome browser and used in downstream analysis.

5.2.1 Dataset properties

The genome of hexaploid sweet potato (*Ipomoea batatas*) and tetraploid *Capsella bursa-pastoris*[50] (shortly CBU) are used for evaluation of methods. The characteristics of these genomes, *i.e.*, SmP interval length distribution, the rate of different type of variants, and genotype distributions are depicted in *Figure 5.1*. The SmP interval length is generated based on the distances between polymorphic sites in the VCF files.

Figure 5.1-A shows how close the variants in these two genomes are. Additionally, it indicates the higher level of heterozygosity in the sweet potato genome since the green line is placed higher than the blue line, and also the negative slope of the sweet potato trend is higher than the CBU genome's trend.

Figure 5.1-B indicates the complexity of variants. Some of the methods for polyploid haplotype assembly are insensitive to non-SNP variants, *i.e.* multinucleotide variants and indels. This plot explains that SNPs are the most common type of variant, however,

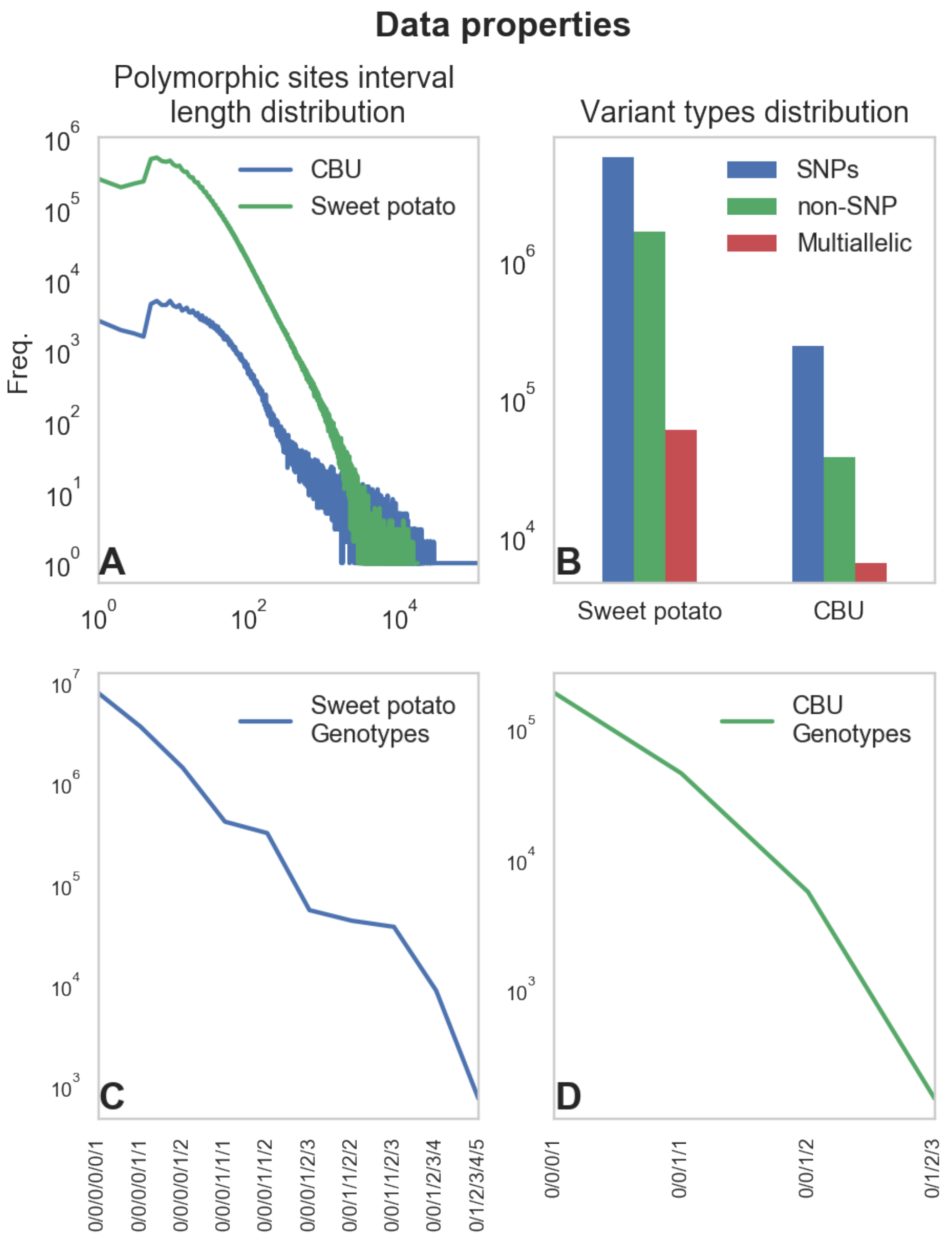


FIGURE 5.1: **Dataset properties of sweet potato and CBU genomes. A)** The interval length distribution of polymorphic sites. Both axes are in log scale. This plot indicates the high number of short intervals and the low number of long intervals. **B)** Different kinds of variants. This plot shows the frequency of the different types of variants, namely SNPs, non-SNP and multiallelic sites. The y-axis is in logarithmic scale. The plot depicts the high number of SNPs in both genomes while the number of the other types of variant is still considerable. Note that, multiallelic variants are a part of non-SNP variants. This plot shows that what proportions of non-SNP variants are multiallelic variants in these genomes.**C, D)** Genotype distribution for sweet potato (left) and CBU (right) genomes. The y-axes are in logarithmic scale. On the x-axes, all possible genotype configurations are shown.

the number of non-SNP variants is substantial. Therefore, in this thesis we use all variants with smaller than 50 base pairs, which we call SmPs. Moreover, we checked how the genotypes are distributed in these genomes.

$$\#\text{reference alleles} \geq \#\text{1st alternative} \geq \#\text{2nd alternative} \geq \ldots$$

The violating genotypes are converted according to the frequency of the alleles; for instance, $\{0, 1, 1, 2\}$ genotype is converted to $\{1, 0, 0, 2\}$ since it is expected to observe more zeros than ones and twos. *Figure 5.1-C,D* shows how the non-biallelic polymorphic sites are distributed.

The mentioned characteristics of the polyploid genomes need to be known for data simulation. To keep all these features in the simulated data, we designed the following pipeline.

For each organism, we selected five scaffold groups of 10kb, 50kb, 100kb, 500kb, and 1000kb length, each of which contains ten scaffolds. The selection process was based on length and coverage so that they represent the majority of scaffolds' properties (*Figure 5.1*). The 50 scaffolds of sweet potato are used for both real and simulated evaluation (due to the availability of semi-true haplotypes, *i.e.*, Roche 454 GS FLX+ pyrosequencer reads) and the 50 scaffolds of CBU are used only for generating simulated data owing to lack of ground truth haplotypes.

5.2.2 Real data

There was no gold standard real dataset for polyploid haplotyping; hence we construct and introduce the following datasets from sweet potato sequencing data[11] (this dataset is available online. See *Appendix B*). We used Illumina short reads for haplotype assembly (see *Figure 5.4* for the properties of sequencing insert sizes and coverage) and the Roche 454 sequencing data for evaluation. The ~1kbp long Roche 454 reads were not used in the *de novo* assembly pipeline. This dataset is used to evaluate the accuracy of assembled haplotypes with the limit of Roche 454 sequence read length (see the read length distribution *Figure 5.3-Right*).

The sequencing error in the 454 reads are more concentrated at homopolymers and the ends; hence, trimming low-quality base calls improves the quality significantly. We set the high threshold of 99.7% (Phred=25) for base quality and 99% for mapping quality (Phred=20). Considering that we only compare variant positions between the haplotypes and the 454 reads (meaning the error in bases should convert one allele to another to cause an error), it is more likely that a match is due to actual sequence identity rather

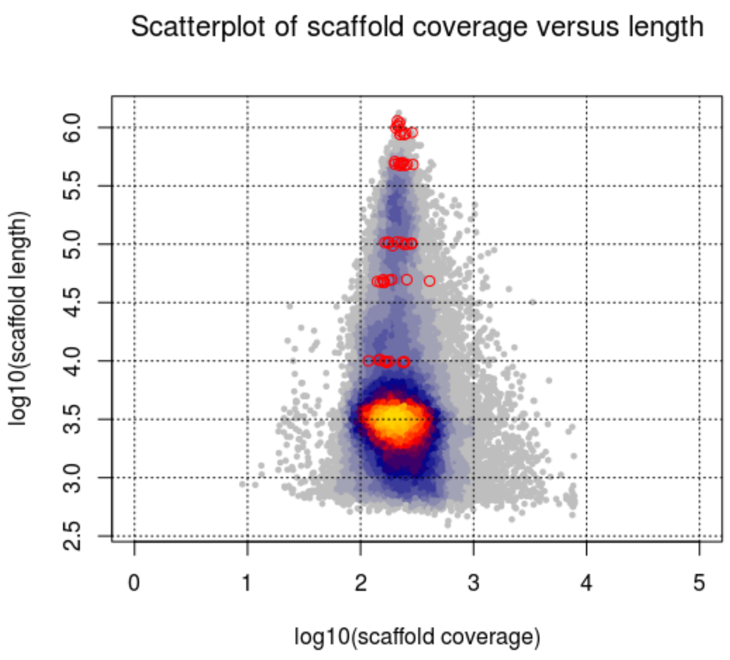


FIGURE 5.2: **Selected scaffolds' properties.** In this plot, each dot depicts one scaffold of the sweet potato genome. These scaffolds were obtained after the scaffolding step of *de novo* assembly. The x-axis shows the log scale of coverage and the y-axis shows the log scale of scaffold lengths. We randomly selected 50 scaffolds of different sizes, namely 10kb, 50kb, 100kb, 500kb, and 1000kb, ten scaffolds each. These scaffolds are used for evaluating with real data and producing the simulated dataset.

than to a sequencing error in the 454 read. For instance, the chance of seeing one error in 70 SmP overlap is less than 19% ($1 - 0.997^{70}$, where 0.997^{70} is the probability of having no error in an overlap of length 70), and it drops exponentially as the length of overlap decreases. Filtering low-quality base pairs and mapped reads gives high-quality ground truth haplotype sequences. For evaluation, the fragments of the mapped 454 reads are extracted. Then, those assembled haplotype segments which contain a shared region with the 454 segments are considered for evaluation. Only one assembled haplotype segment out of six is assigned to the corresponding 454 fragment for assessment based on similarities and dissimilarities (*Eq.4.5*).

5.2.3 Simulated data

By usingutilizing the real scaffolds and their variants, the properties of real data such as interval length and complexity of the polymorphic sites are conserved. Based on these scaffolds and their randomly shuffled genotypes, P haplotypes are generated. After that, each haplotype is used as a reference for the read simulator tools EAGLE [51] and pIRS [52]. We constructed two datasets, one small and one large. The small data set is generated to test all of the four mentioned state-of-the-art-methods. *Figure 5.4* depicts the

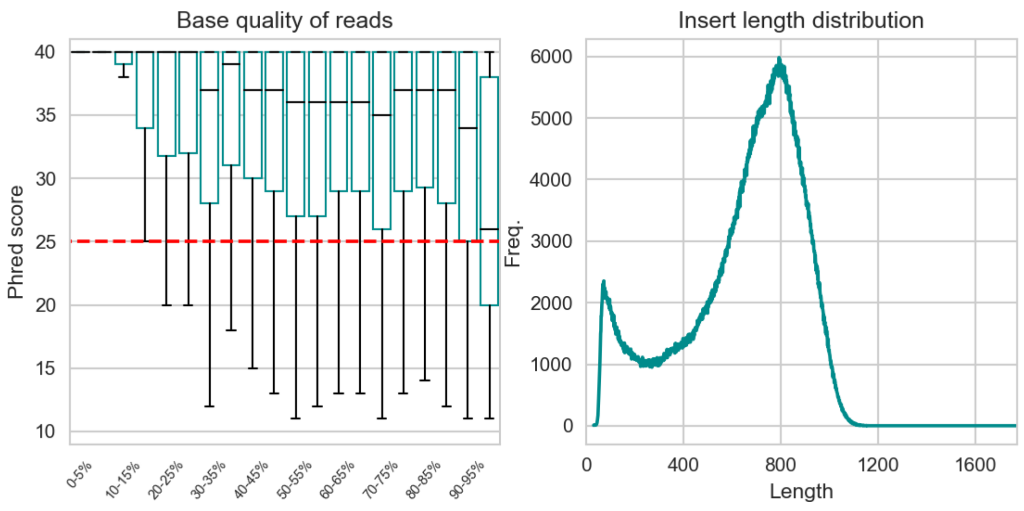


FIGURE 5.3: **Base quality and length distribution of Roche 454 reads. Left)** Each 454 read is divided into 20 equal size segments. The box plot shows the base quality distribution and the red line indicates the high threshold we set for filtering the bases on quality. **Right)** The Roche 454 length distribution. Maximum length is 1771bp

.

features of the small simulated dataset, and its comparison to the real data characteristics. ***HapCompass*** and ***SDhaP*** were not able to accomplish the haplotyping of long scaffolds. Therefore, to test ***H-PoP*** and ***Ranbow*** in more details, we generated a large dataset, consisting of four insert sizes. Moreover, we collected these four data sets in one larger dataset as well (see *Figure* 5.5). The reads from different haplotypes are then collected in one BAM file. The BAM file and the corresponding VCF file are used as a simulated dataset.

5.3 Evaluation

5.3.1 All methods

In this section, the performance of ***Ranbow*** is evaluated and compared to the state-of-the-art methods. All of these methods are applied on real and simulated data sets. *Figure* 5.6 shows the general performance of different methods.

Match-mismatch plots: The first and second columns of *Figure* 5.6 show the result on real and simulated data, respectively. The scatter plots in the first row, which we call "*match-mismatch plots*", illustrate the accuracy of methods. One dot in these plots is one assembled haplotype that is compared to the ground truth. The colors indicate the frequency. The x-coordinate (y-coordinate) of a dot indicates how many alleles are assembled correctly (incorrectly). Since the aim of haplotyping is obtaining longer and

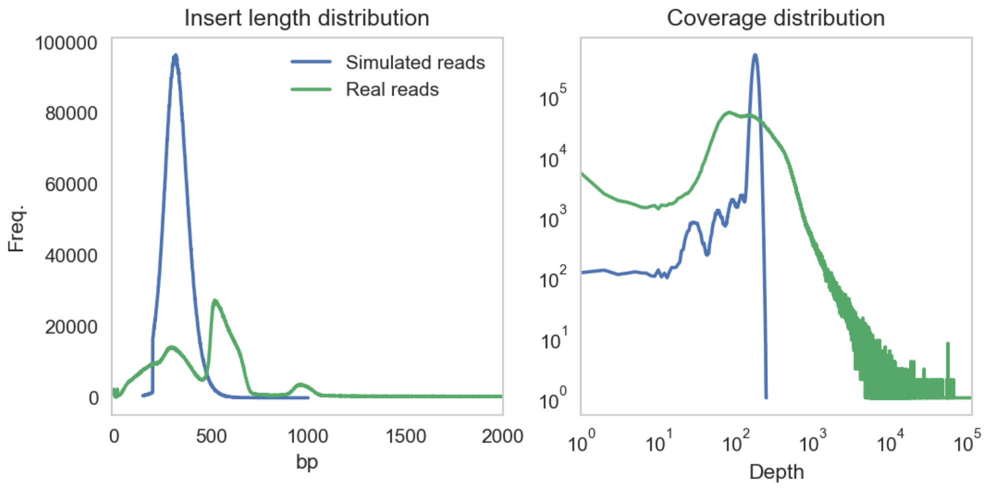


FIGURE 5.4: **Dataset properties for sweet potato genome. Left)** Insert size distribution. The real data contains five different libraries with different insert sizes, namely 350bp, 550bp, 950bp, 20k, and no size selection. For simulated data, we generated inserts of 350bp from the selected scaffolds with 30x coverage for each haplotype. **Right)** Coverage of selected scaffolds for real and simulated data. The x-axis shows base coverage, and the y-axis depicts frequency. In the real dataset, the base coverage varies in a wide range up to 10k while the simulated data has the peak at 180x. This discrepancy is caused by the presence of repeats in the genome.

more accurate sequences, a longer x-coordinate and shorter y-coordinate are desired. The flatter the slope of a line, the higher the accuracy of the method it belongs to.

Figure 5.6-A presents the match-mismatch plot for sweet potato real data. The x-axis is the number of matches between the alleles in Roche 454 reads and assembled haplotype, while the y-axis depicts mismatches between them. Since the length of Roche 454 reads is limited, the maximum size of the overlap between assembled haplotypes and these reads is limited as well. The lines depict linear fits to the dots of each method. *Figure 5.6-B* is the match-mismatch plot for simulated data generated from the comparison of the assembled haplotypes and the genomes' simulated haplotypes. Since in simulated data the six haplotypes are available, there is no limit for the size of overlaps. These plots show that ***Ranbow*** performs best with respect to accuracy, as illustrated by the lower slopes of the fitted lines.

Execution times: The lower plots show the execution times for different scaffold size groups (10 scaffolds per five scaffold length groups, 10×5 scaffolds in total). The y-axes are in logarithmic scale. ***Ranbow*** is at least one order of magnitude faster than the other methods for real data in all scaffold size groups (*Figure 5.6-C*), and it is several times faster for simulated ones (*Figure 5.6-D*).

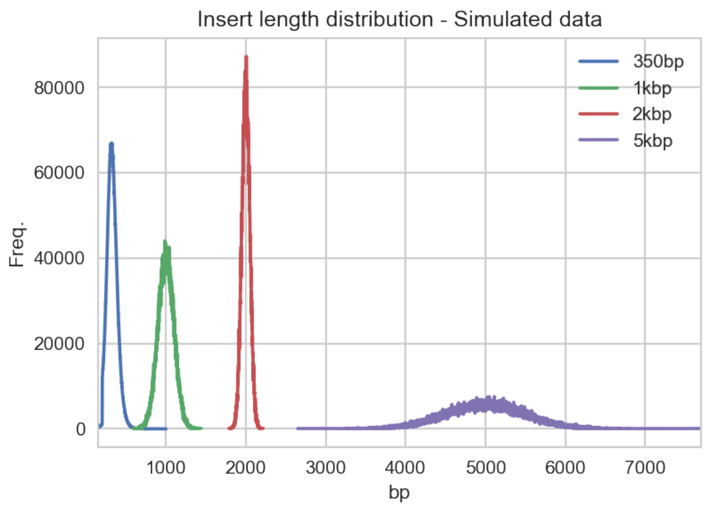


FIGURE 5.5: **Insertion length distribution for simulated dataset of CBU genome.** Four 100 bp paired-end read libraries with the insert sizes of 350bp, 1kbp, 2kbp and 5kb are generated by EAGLE (Enhanced Artificial Genome Engine)[51]. EAGLE generates reads and converts them to alignments. It is designed to simulate the behavior of Illumina's Next Generation Sequencing instruments. For each library, the coverage for every haplotype is 40x.

Accuracy and haplotype length: To investigate the performance of all methods in more detail, we split the haplotypes based on their lengths into five and four categories for simulated and real data respectively: very short, short, medium, long, and very long. The real data does not contain the very long category since the evaluation of real data is done by Roche 454 reads, which is limited in size. Here, we compare the accuracy distribution of the haplotypes in different groups. *Figure 5.7* shows if there is any dependency of the assembled haplotype length and their accuracy, and how ***Ranbow*** performs in different haplotype length categories. Recall that, the accuracy is defined as:

$$\frac{\#matches}{\#matches + \#mismatches} \tag{5.1}$$

The results on real and small simulated datasets are shown as box plots in *Figure 5.7*. These plots indicate, independent of the size of the assembled haplotype, that ***Ranbow*** outperforms in terms of accuracy.

Haplotype length: Although the box plots in *Figure 5.7* show the length distribution in each category as well as the accuracy, to compare the haplotype length more precisely, we depict the histogram of assembled haplotype length in real data (*Figure 5.8*). The y-axis shows the count on a logarithmic scale. This plot shows ***Ranbow*** and ***H-PoP*** outperform the other methods in terms of assembled haplotype length.

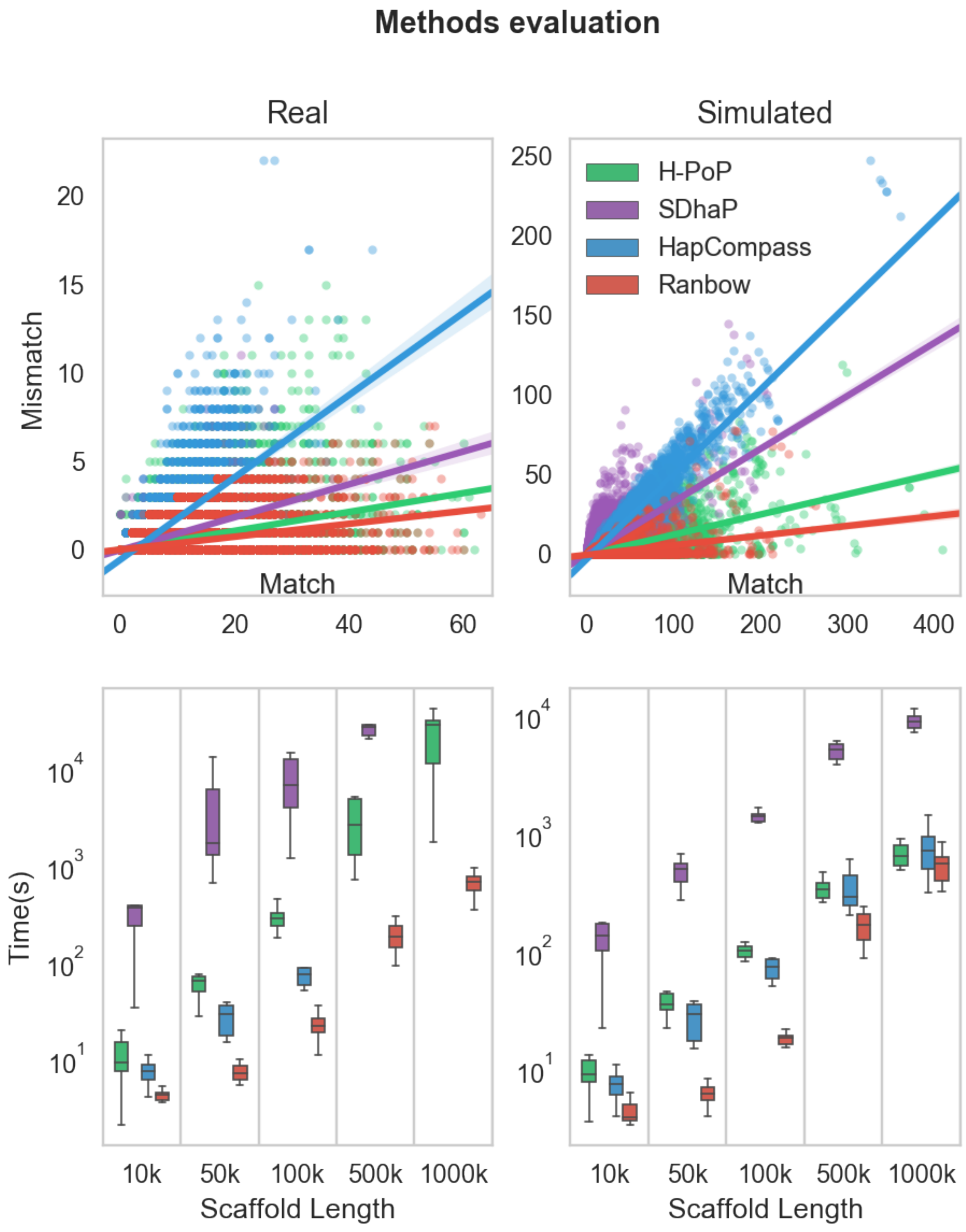


FIGURE 5.6: **Comparison of all methods on sweet potato real and simulated datasets**. A, B) match-mismatch plots for sweet potato real and simulated datasets C, D) The execution times for different scaffold size groups

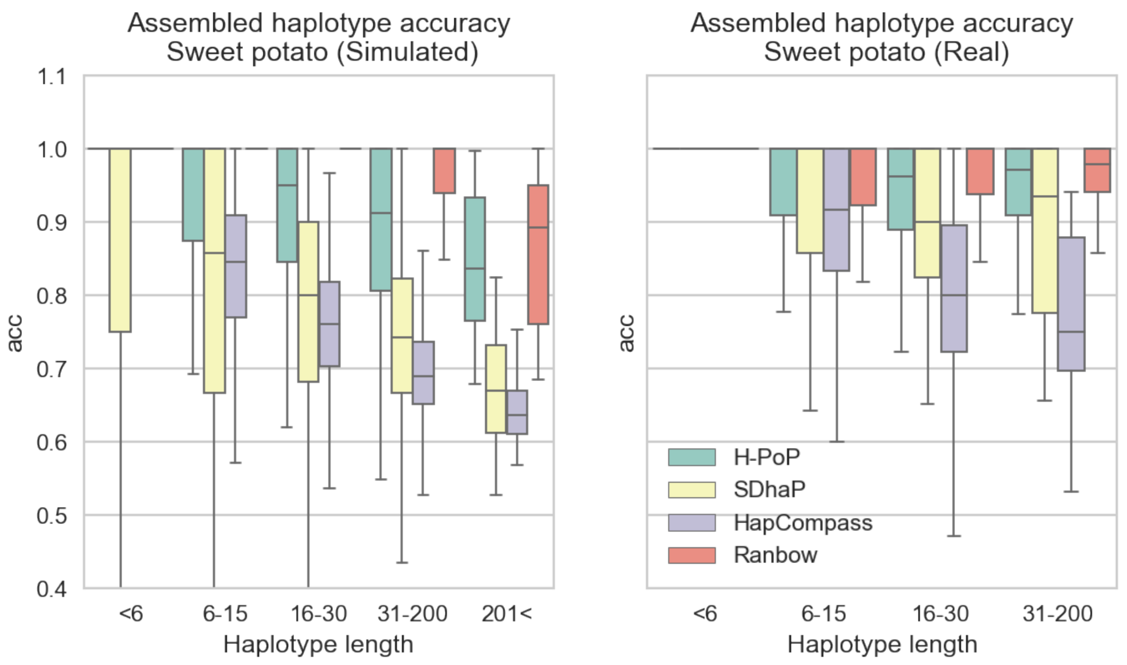


FIGURE 5.7: **Comparison of the methods regarding accuracy on real and simulated sweet potato data. Left)** The assembled haplotypes of simulated data are grouped into five groups according to their size. **Right)** Since the size of real ground truth data is limited, the real data is grouped into four categories.

5.3.2 ***H-PoP*** vs ***Ranbow***

To investigate the performance of ***H-PoP*** and ***Ranbow*** in more detail, we produced four simulated libraries, namely 350bp, 1kbp, 2kbp, and 5kbp for each organism (see the details of insert size for CBU genome in *Figure* 5.5). Moreover, all of the reads of these four libraries are collected in one extra library, which is depicted as 'All' in the plots.

Match-mismatch plot: *Figure* 5.9 shows the match-mismatch plots for ***H-PoP*** and ***Ranbow*** for different insert sizes. These plots indicate that ***Ranbow*** is more accurate and assembles longer haplotypes; the dots are closer to the x-axis. Moreover, its performance increases when different sequencing libraries are integrated, which is mostly the case for sequencing of new organisms. Comparing *Figure* 5.9-*All-Ranbow* and *Figure* 5.9-*All-H-PoP* depicts how well ***Ranbow*** is usingutilizing the four insert sizes to produce longer haplotypes but with lower mismatches. These plots also depict the importance of using different insert sizes for haplotype assembly.

The same principle applied to sweet potato data: *Figure* 5.10 indicates the better performance of ***Ranbow*** in terms of length, number of assembled haplotypes, and their accuracy. ***Ranbow*** shows higher accuracy than ***H-PoP***, *i.e.* the dots are closer to the x-axis in all insert sizes. However, the assembled haplotypes are not as good as the ones assembled from CBU genome owing to higher heterozygosity and ploidy. The higher

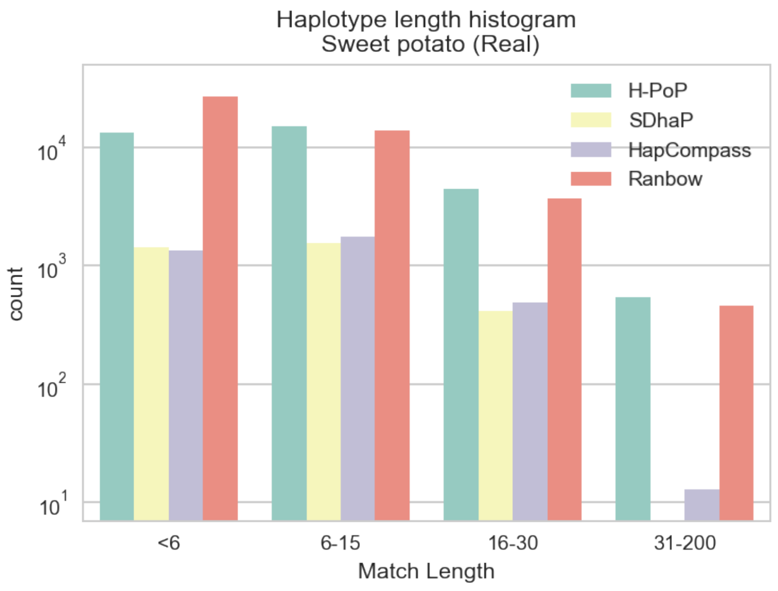


FIGURE 5.8: **Comparison of all methods in terms of assembled haplotype length on sweet potato real data.** This plot shows ***Ranbow*** and ***H-PoP*** perform better in terms of assembled length.

heterozygosity causes higher allele connections resulting in longer haplotypes, and the greater ploidy may introduce higher switch errors that result in more error which is shown in *Figure 5.10*.

Accuracy: We proceed to show that the comparison of methods does not change when looking at particular scaffold lengths. For this purpose, we categorized the performances by scaffold lengths in *Figure 5.11*. In this figure, each dot represents the average accuracy of haplotypes belonging to each insert size. This plot is generated from CBU data. In *Figure 5.11*, the x-axis shows five different categories of scaffold length, the y-axis is the average accuracy measure, each dot is the average accuracy of 10 scaffolds phased from the same insert size. This plot shows ***Ranbow*** outperforms ***H-PoP*** in all categories; the red dots are always located above the green ones except the annotated green dot. The annotated green dot is the accuracy of ***H-PoP*** on 350bp insert size and the corresponding red dot for the same insert size obtained from ***Ranbow*** method is pointed out as well. This plot shows ***Ranbow*** performs better for all insert sizes in all scaffold length categories.

Haplotype length: We compared ***Ranbow*** and ***H-PoP*** in terms of the number of connected alleles which are correctly assembled into haplotypes. In other words, the mismatches are ignored in this comparison. *Figure 5.12* shows this comparison. Each pair of violin plots is obtained from one insert size. The y-axis depicts the number of matches in the assembled haplotype. The width of the violin plots shows the histogram of the number of assembled haplotypes within the groups. Except for the *350bp* group,

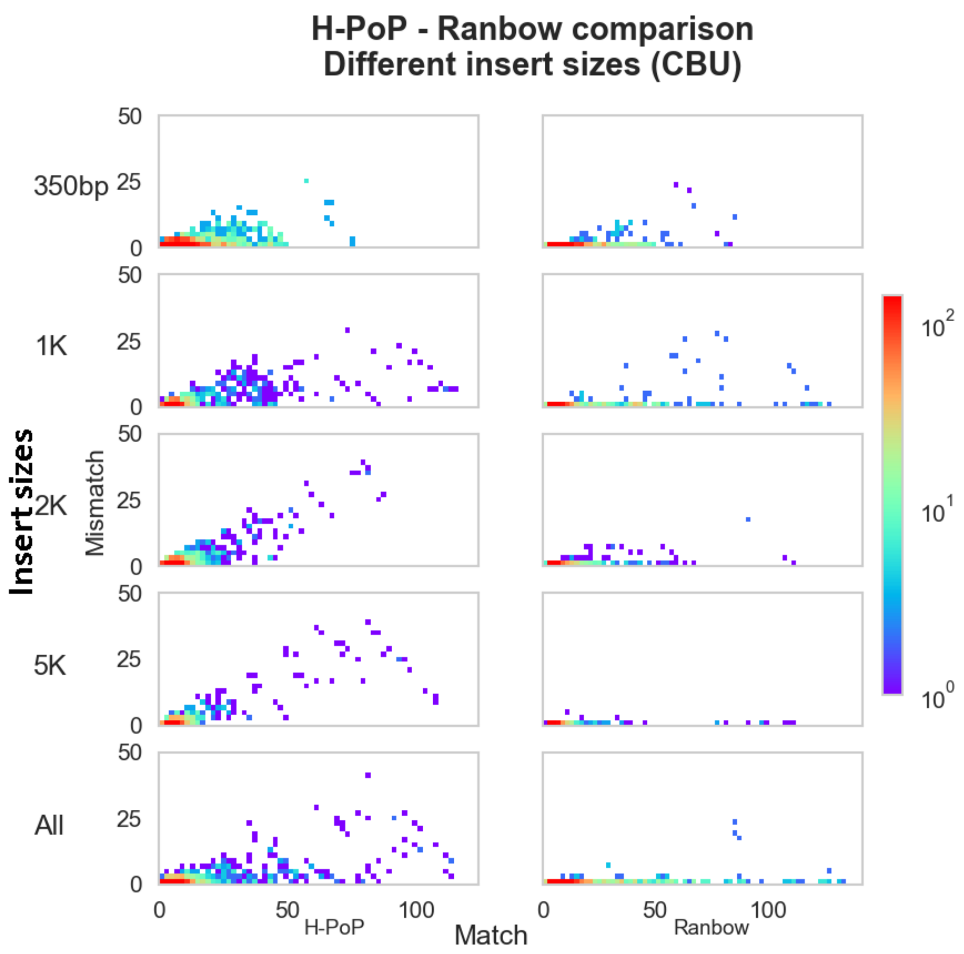


FIGURE 5.9: **Comparison of *H-PoP* and *Ranbow* on different insert sizes for simulated dataset of CBU genome**. The match-mismatch plots show the effect of different insert sizes on the performance of the methods. The colors indicate the frequency. ***Ranbow*** performs best in almost all conditions; the dots are closer to the x-axis, and also the number of matches is longer in the right plots. Moreover, the combination of all insert sizes improves the result dramatically (lower plots). There are a good number of assembled haplotypes containing very low number of mismatches in the lower right plot.

in which ***H-PoP*** performs better, ***Ranbow*** outperforms in all groups including the collection of all insert sizes, which is depicted in *all*.

Error correction and reconstruction rates: Another measure for comparing methods is the rate of error correction. This measure is designed using the Minimum Error Correction model (explained in *Section 3.2*). It is calculated based on the assignment of reads to the haplotypes. The number of allele corrections needed to make the reads compatible to the haplotypes they assigned to is calculated. The error correction rate can be used for comparing methods when no ground truth haplotype is available. Since we utilized Roche 454 reads for real data and the reference haplotypes for simulated data as the ground truth for evaluation, the accuracy can be easily calculated and there

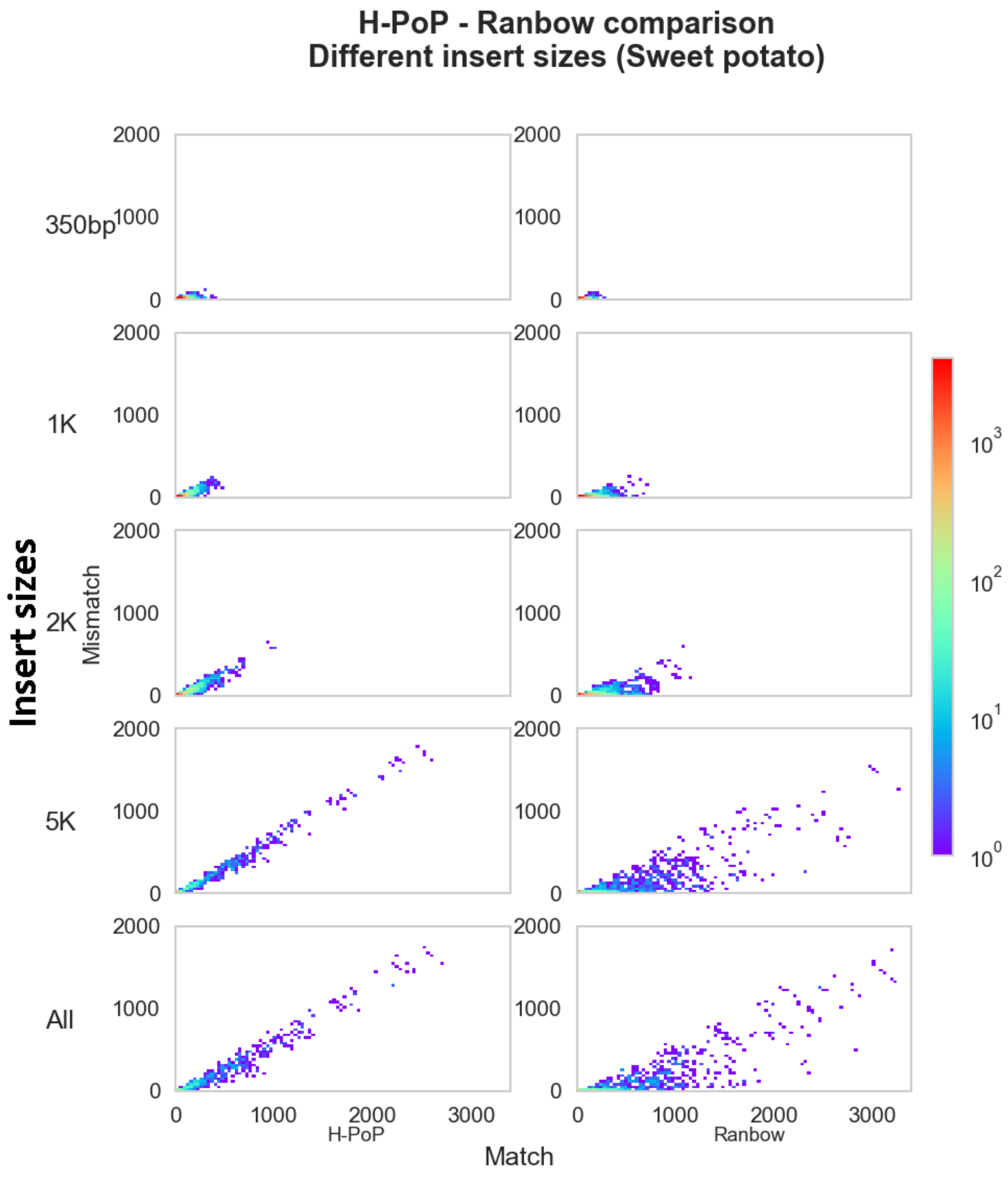


FIGURE 5.10: **Effect of different insert sizes on the performance of *H-PoP* and *Ranbow* on sweet potato simulated data.** This figure shows that ***Ranbow*** performs better not only on various insert sizes, but it also outperforms when all libraries are merged into one dataset. The colors indicate the frequency.

is no need for extra measures. Nevertheless, we calculated the error correction measure for our haplotype assembly as well. It is worth noting that although ***Ranbow*** is not designed to optimize this measure, it produces a very low error correction rate. *Figure 5.13*, which looks like a match-mismatch plot, shows the distribution and the average rate of corrected alleles in the assembled haplotype. In other words, this figure shows

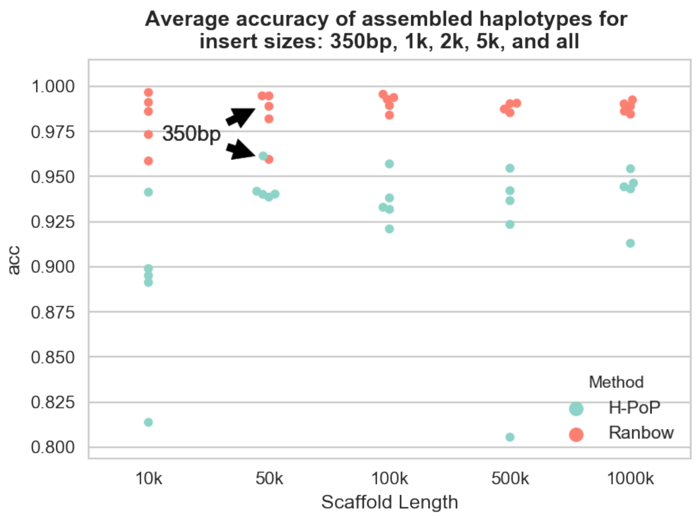


FIGURE 5.11: **Comparison of *H-PoP* and *Ranbow* regarding accuracy on CBU simulated dataset.** ***Ranbow*** results in more accurate haplotypes in all groups with various scaffold lengths.

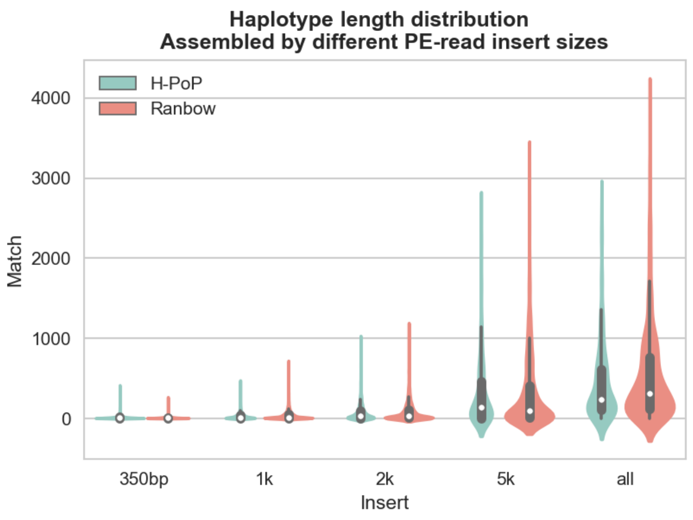


FIGURE 5.12: **Matching length and frequency distribution of assembled haplotypes by different insert size on simulated sweet potato data.** In this plot the mismatches are removed; hence it shows the number of correctly connected alleles.

how many inconsistencies are found between the haplotypes and the reads they are assembled from. The x-axis shows the length of haplotypes while the y-axis indicates the number of inconsistencies between the haplotypes and its reads. The closer the dots to the x-axis the lower the error correction rate assigned to the corresponding scaffold. The reconstruction rate is calculated as $1 - error\ correction$; it indicates how many correct alleles from the reads were involved in assembling the haplotypes. The average reconstruction rates are shown in the title of the plots.

The reconstruction rate of ***Ranbow*** on real and simulated date are 0.9997 and 0.9996 respectively. However, ***Ranbow*** does not involve all of the reads in the assembly. It

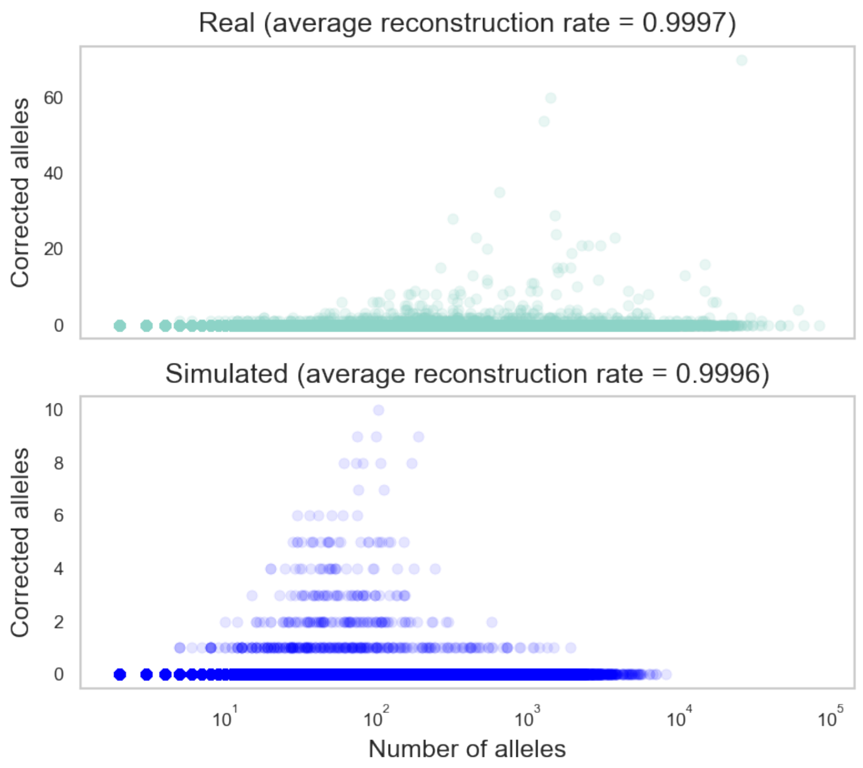


FIGURE 5.13: **Scatter plots of corrected alleles in assembled haplotypes.** Each haplotype is assembled from a collection of fragments that may contain a few disagreements with each other. The average reconstruction rate, *i.e.* $1 - \textit{error correction}$ for both real and simulated datasets are more than 99.9%

filters the reads which may not belong to the region and assembles the haplotypes from a set of highly confident reads. Since not all of the generated reads are considered to calculate the error correction rate, our approach does not follow the MEC model entirely. Thus, we skip the comparison of ***Ranbow*** and other methods based on MEC model and only report the rate of error correction among selected fragments by ***Ranbow***. Recall that MEC model (Minimum Error Correction model) tries to cluster and correct errors in reads such that all reads in each cluster are compatible with each other (*Section 3.2*).

5.3.3 Comparison of the results on real and simulated datasets

To investigate the performance of ***Ranbow*** on real and simulated data, we compared them in terms of number of haplotypes in a phased region (**Number of Inferred Haplotypes**) (*Figure 5.14*), and the rate of connectivity in multiallelic sites (*Figure 5.15*) in the following paragraphs.

Number of Inferred Haplotypes: Number of inferred haplotypes is the number of haplotypes in a phased region. This number could be P as the ploidy of the organism or fewer than P. The latter case could be due to the characteristics of the genome in the region or owing to the failure in sequencing (see *Section 4.3.4* for more details on how ***Ranbow*** deals with these regions.) We investigate Number of Inferred Haplotypes for real and simulated data. It is worth noting again that the underlying scaffolds and variants for both real and simulated reads are the same. Moreover, it is infeasible to check if the Number of Inferred Haplotypes is correctly called since there are no clear boundaries for the regions with different Number of Inferred Haplotypes. On one hand, the whole scaffold is the Number of Inferred Haplotypes equal to P. On the other hand, by putting the boundaries between the polymorphic sites such that each interval contains only one polymorphic site, the Number of Inferred Haplotypes would be the same as the size of the genotype for the site. Hence, the same genome can be annotated with several sets of true Number of Inferred Haplotypes according to the boundary selection.

In *Figure 5.14* we investigate the Number of Inferred Haplotypes from both real and simulated reads. We divided our analysis into four groups, with or without gaps, in nucleotide space and coded allele space. These plots indicate most of the regions in both datasets are phased into P haplotypes. The sum of the regions with P haplotypes is far higher than the simulated one due to the long insert sizes; however, the plots with removed gaps show that many of the alleles in these haplotypes are not called. The regions in P-ploid genomes contain fewer than P haplotypes when a piece of chromosome is identical in two or more copies. ***Ranbow*** calls these regions as well, whose properties are depicted in *Figure 5.14*.

Assembly of non-SNP sites: Additionally, we investigate the performance of ***Ranbow*** on multiallelic and non-SNP polymorphic sites (non-SNPs). We checked the effect of these variants on haplotype assembly and how accurately they are assembled. For this purpose, the number of non-SNP sites which are accurately called in each haplotype is checked. *Figure 5.15* illustrates the distribution of the accuracies in assembled haplotypes on non-SNP polymorphic sites of the sweet potato simulated dataset. The accuracy rate of 0.79 indicates that most non-SNP alleles are integrated correctly into the haplotypes. However, the precision of integrating this type of variant is lower than for SNP variants; nevertheless, more of the dots sit in the right lower side of the plot and are close to the x-axis, which shows that ***Ranbow*** performs well in integrating these variants into the haplotypes.

5.3.4 Usability

Finally, we compared the methods from the usability point of view in *Table 5.1*. We compared these methods based on their demand on pre and postprocessing, and also

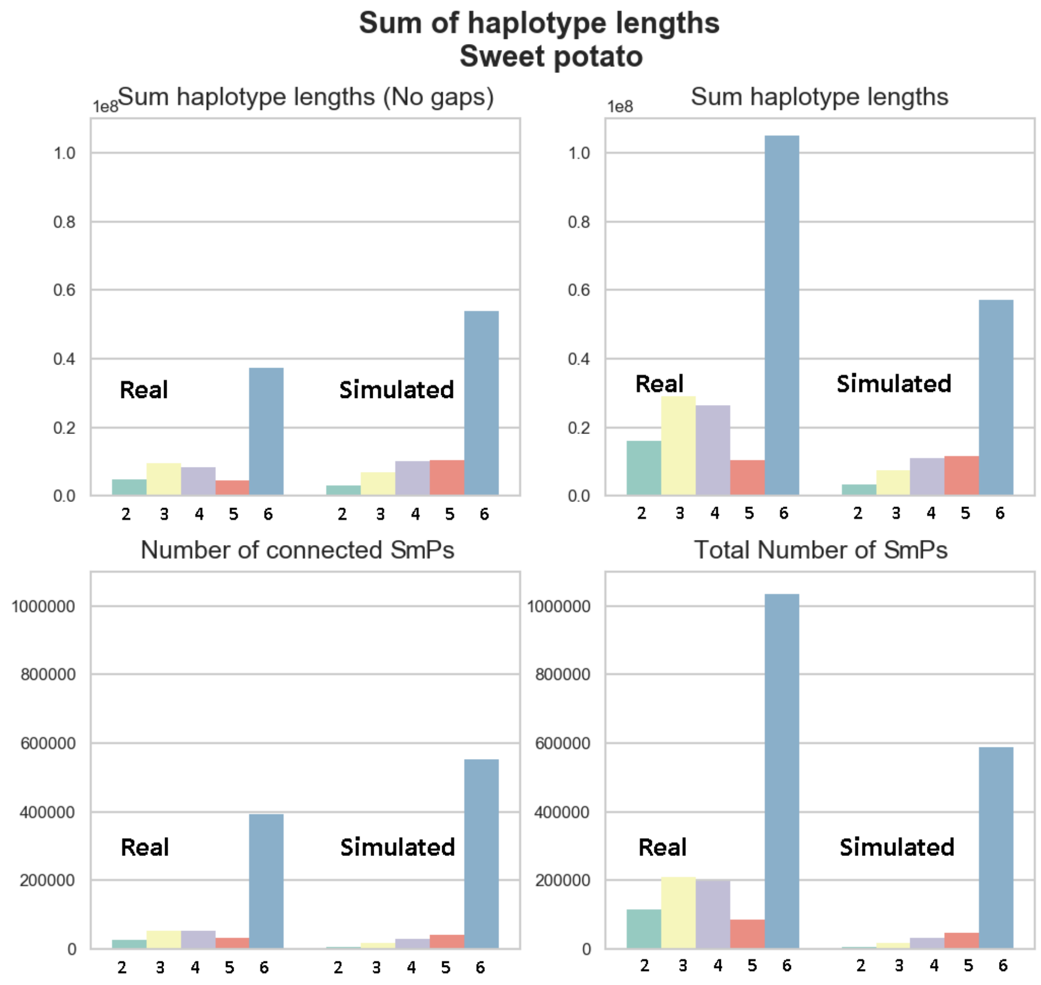


FIGURE 5.14: **Frequency of Number of Inferred Haplotypes in the phased regions.** The upper plots show the haplotype length in nucleotide space, and the lower plots show the same in coded allele space. In the left plots, the missing data is removed from the haplotypes. These plots indicate the number of gaps in the produced haplotypes in the real data set, due to the long insert size sequencing libraries used for real data sets. Moreover, these plots show both real and simulated data assemble almost the same number of haplotypes with the same number of copies.

their non-SNP and multiallelic variants handling. ***Ranbow*** and ***HapCompass*** accept BAM and VCF format as input and produce aligned haplotypes as in a BAM format. There is no need for pre or postprocessing to make the data ready for applying the algorithm and using the result for downstream analysis. However, ***H-PoP*** and ***SDhaP*** accept fragment format. In this format the alleles are coded into numbers and their connectivity via reads are shown as a sequence of numbers. As output, these methods produce a fragment file as well. Therefore, to apply these methods on real data, both pre and post-processing steps need to be carried out to make these tools ready for real applications. Moreover, ***HapCompass***, together with ***Ranbow***, are the only methods that can handle non-SNP sites. ***SDhaP*** cannot handle non-SNP polymorphic sites if the

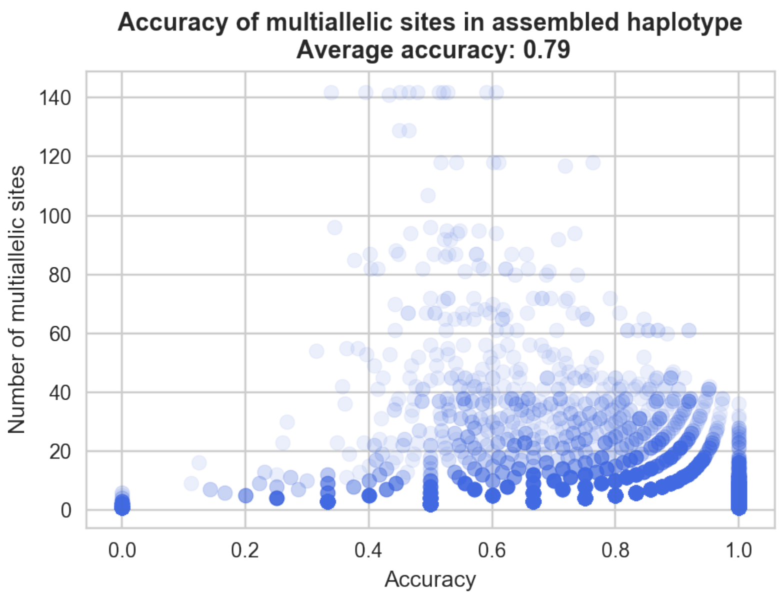


Figure 5.15: **Distribution of the accuracies in assembled haplotypes on multiallelic polymorphic sites of sweet potato simulated dataset.** Each dot shows one assembled haplotype. On the x-axis, the accuracy of assembled multiallelic polymorphic sites is depicted while the y-axis shows the number of multiallelic sites. This plot shows that, as the number of multiallelic sites increases, the accuracy rate drops, but still, there are a good number of long haplotypes with very high accuracy, and the average accuracy of all dots is 79%.

number of alleles is higher than four; therefore, this restriction needs to be considered in the preprocessing step for this method. ***H-PoP*** does not integrate more than two SmP alleles in the haplotype assembly, it converts them to either the reference or first alternative alleles (*Table* 5.1).

	Ranbow	**H-PoP**	**SDhaP**	**HapCompass**
Additional pre- and postprocessing	No	Yes	Yes	No
non-SNP handling	Yes	∗	∗	Yes

∗ There are restrictions on handling these sites.

Table 5.1: Usability of methods

5.4 Conclusion and Discussion

We applied ***Ranbow*** and three state-of-the-art methods on hexaploid sweet potato sequencing data. The results were then evaluated by recruiting long Roche 454 reads. This dataset can be used as a gold standard for polyploid haplotypers. We also generated two simulated datasets inspired by sweet potato and tetraploid CBU genomes. In order to compare the performance of different methods on short sequence reads, we generated four insert size libraries, in the 350bp to 5kbp range, with 100bp single-end read length.

The performances of all methods were evaluated on real and simulated datasets concerning accuracy, length of assembled haplotypes and execution time. ***Ranbow*** performs best in all measures. Moreover, ***Ranbow*** was more than one order of magnitude faster than the other methods, which was enough to be applied on the whole sweet potato genome for several iterations. The iterations were used to useutilize haplotyping to improve the scaffolding step of *de novo* assembly (details are explained in *Chapter 6)*. With this speed each iteration took less than two days on the processor farm, which means it is infeasible for the next best method to be applied to the same data. Furthermore, ***Ranbow*** resulted in 40% of phased haplotype regions in the sweet potato genome.

Through emerging third generation sequence technologies and decreasing cost of obtaining long reads, one avenue for future work would be scaling the current method or designing new methods for usingutilizing this type of data. The other important direction would be combining different sequencing technologies. As an example, the SMRT technologies provide erroneous long allele connections while the errors can be detected and fixed by short read data. Furthermore, the integration of base quality seems necessary, especially for using only SMRT reads. Linked read technology (10X Genomics) helps to accurately call the variants but it suffers from the possibility of combining two haplotype segments into one read. All in all, not one sequence technology can thoroughly solve the haplotyping of the genome - specifically polyploid genomes - at this time; thus, the combination of methods depending on the characteristics of the target genome seems to be a more reasonable direction for future research. The haplotyping could be integrated into the genome assembly and scaffolding to obtain more accurately assembled genomes.

Chapter 6

An application of haplotype assembly: Haplotype aware *de novo* assembly of hexaploid *Ipomoea batatas*

This chapter describes a collaborative work of the Max Planck Institute for Molecular Genetics with the Chinese Academy of Sciences. In this project, we assemble the sweet potato genome in collaboration with Jun Yang, an independent researcher in our lab, and the sequencing core facility of the Max Planck Institute for Molecular Genetics. DNA material was transferred from China and sequenced in house. The initial *de novo* assembly was done in the sequencing core facility by Heiner Kuhl and provided to us for further analysis and improvement. This collaboration resulted in a report published in Nature Plants[11]. In this chapter, after explaining the *de novo* assembly part of this project, we explain a novel method for improving the assembly, more specifically the scaffolding part, called haplotype-aided scaffolding. The proposed method relies on the assembled haplotypes and the connections between scaffolds which are inferred through the reads mapped to these haplotypes. After obtaining a new assembly, the haplotyping can be repeated. This makes an iterative cycle between the assembly and haplotyping which improves the assembly stepwise. More detail is provided in the following sections.

6.1 Introduction

With a consistent global annual production of more than 100 million tons, as recorded between 1965 and 2014 (FAO), the sweet potato *Ipomoea batatas*, is an important source of calories, proteins, vitamins and minerals for humanity. It is the seventh most important crop in the world and the fourth most significant crop in China. In periods of shortages of basic cereal foods, *I. batatas* is frequently served as the main food source for many Chinese. It rescued millions of lives during and up to three years after the Great Chinese Famine in the 1960s and was subsequently raised as a main guarantor of food security in China.

The sweet potato has a complex and very difficult to assemble genome with a genome size of about 4.4 Gb. It is a hexaploid organism with haploid number 45 (n=45), *i.e.* number of the chromosomes in the gamete cells, and 15 monoploid chromosomes (x=15) which results in $2n = 6x = 90$. It has a composition of two B1 and four B2 component genomes (B1B1B2B2B2B2), as predicted by genetic linkage studies using RAPD and AFLP markers[53, 54]. This nomenclature shows a hybridization of a diploid B1 and a tetraploid B2 organism followed by a duplication.

The sweet potato is highly polymorphic. Our initial sequencing of the *I. batatas* genome revealed that the distance between adjacent polymorphic sites is roughly one tenth of the distance in the human genome. Based on previous statistics, there are approximately 14 million polymorphic sites in the estimated 700∼800 Mb monoploid genome of *I. batatas*. This means that, on average, one read (100∼150bp length) from Illumina sequencing will cover 2∼3 polymorphic sites. This density of such sites should permit phasing the *I. batatas* genome, employing cost-effective Illumina sequencing with paired-end libraries.

While the high heterozygosity of *I. batatas* makes genome assembly more challenging, it simultaneously makes haplotyping easier. This property suggests that the high heterozygosity could benefit in phasing the genome to fairly long haplotypes and, furthermore, to employ these haplotypes for improving the genome assembly. In this chapter, we present the pipeline designed for *de novo* assembly of sweet potato genome. Moreover, we detail how phased regions could potentially advance the genomes' *de novo* assembly.

6.2 Pilot project

A newly bred carotenoid-rich cultivar of *I. batatas*, Taizhong6 (China national accession number 2013003) from China, was used for genome sequencing at the MPI core sequencing unit. During the genome survey stage, three sequencing libraries were constructed and sequenced on Hiseq2500 and GS FLX+ platforms (*Table 6.1 - A500, A1kb and A454*). After a preliminary genome assembly, we obtained ∼166k scaffolds with the N50 of ∼60kb. Thereafter, the reads were mapped to the scaffolds, and the PCR duplicates were removed. Mapped reads were employed for variant calling to meet the requirements for haplotyping. A small sample of scaffolds (n=75) was chosen on the basis of scaffold length and the coverage of informative reads for haplotyping, *i.e.*, reads contain at least two sequence variants (*Figure 6.1*). The haplotyping on selected scaffolds was carried out using ***Ranbow*** in order to phase the genome into up to six haplotypes, into triploid, tetraploid, pentaploid, and hexaploid regions.

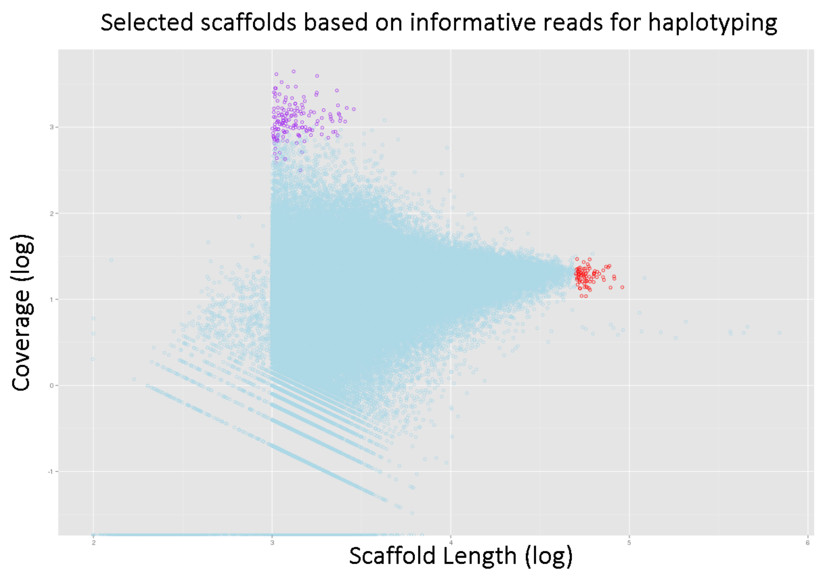


Figure 6.1: **Selected and highly covered scaffolds.** The selected scaffolds are shown in red. These scaffolds were chosen on the basis of scaffold length and informative read coverage. Total number of scaffolds is ∼166k. The reason for selecting such a subset was anticipating more scaffolds closer to this region by more sequencing since they are sufficiently long and covered by reads. The purple dots illustrate very highly covered scaffolds, which are more likely to be repeat regions or small assembled haplotypes which are not integrated in the longer scaffolds due to the heterozygosity. It was predicted that these scaffolds will be mapped to the longer ones by enhancing the quality of assembly.

Library	Platform	Sequencing type	Insert size	QC-passed reads	Mapped rate*
A500	Hiseq 2500	PE100	350 bp	345333619	94.66%
A1kb	Hiseq 2500	PE100	950 bp	190263659	95.07%
L500	Nextseq 500	PE150	550 bp	837098772	94.70%
MP	Nextseq 500	PE150	No size selection	694919486	91.65%
AMP	Hiseq 4000	PE100	20kb	316973015	95.67%
A454	GS FLX+	SE (up to 1kb)	-	3385694	98.91%
Total	-	-	-	2387974245	93.97%

* Map against preliminary assembly

Table 6.1: Sequence library characteristics

The coverage of these regions led us to estimate the minimum coverage needed for haplotyping the whole genome. This was inferred according to the peak coverage of hexaploid regions in *Figure 6.2*. The observed coverage for the regions with fewer copies of haplotypes are lower than the coverage peak of hexaploid regions (*Figure 6.2*). This suggests the sequencing coverage is not enough for the regions predicted as non-hexaploid. *Figure*

6.2 shows that 40-fold monoploid genome coverage, at minimum, is needed to phase the genome into six haplotypes.

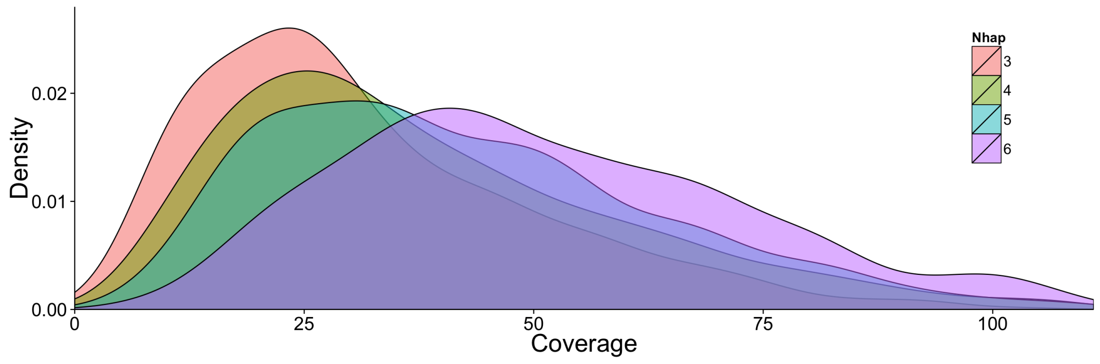


FIGURE 6.2: **Coverage distribution of genomic regions phased into triploid, tetraploid, pentaploid, and hexaploid during genome survey.** Based on the genome survey data and primary assembly, genome regions have been phased into up to six haplotypes. The coverage of each phased region and number of phased haplotypes is summarized here. The peak coverage around 40 in hexaploid indicated the minimal sequencing depth requirement for haplotyping of hexaploid genome. The peak coverage shifting from triploid to hexaploid demonstrated the insufficient sequencing depth in the genome survey stage.

Moreover, short inserts are preferential for haplotyping since ***Ranbow*** sits on the regions with at least six unique sequence patterns; the smaller the insert, the higher the chance of connecting consecutive polymorphic sites, the higher the chance of finding six unique sequence patterns. On the other hand, longer insert sizes of paired-end reads are beneficial for scaffolding. Considering these facts, new libraries were sequenced on the Nextseq500 platform to meet the estimated coverage requirement (*Table 6.1-L500*), and additionally gel-free mate-pair and 20k mate-pair libraries were also sequenced to improve scaffolding (*Table 6.1-MP and AMP*. The insert size distributions of these sequence libraries are shown in *Figure 6.3*).

6.3 Preliminary assembly

6.3.1 Initial assembly of consensus genome

A heterozygosity-tolerant assembly pipeline, combining de Bruijn[55] and OLC[56] (overlap layout consensus) graph strategies, was employed to carry out the hexaploid genome assembly of *I. batatas*, using error corrected Illumina reads. A total length of ~870Mb, mainly representing the monoploid genome, was assembled using this pipeline. We found the assembled endophyte *Bacillus pumilus* genome as the largest scaffold being 3.7Mbp

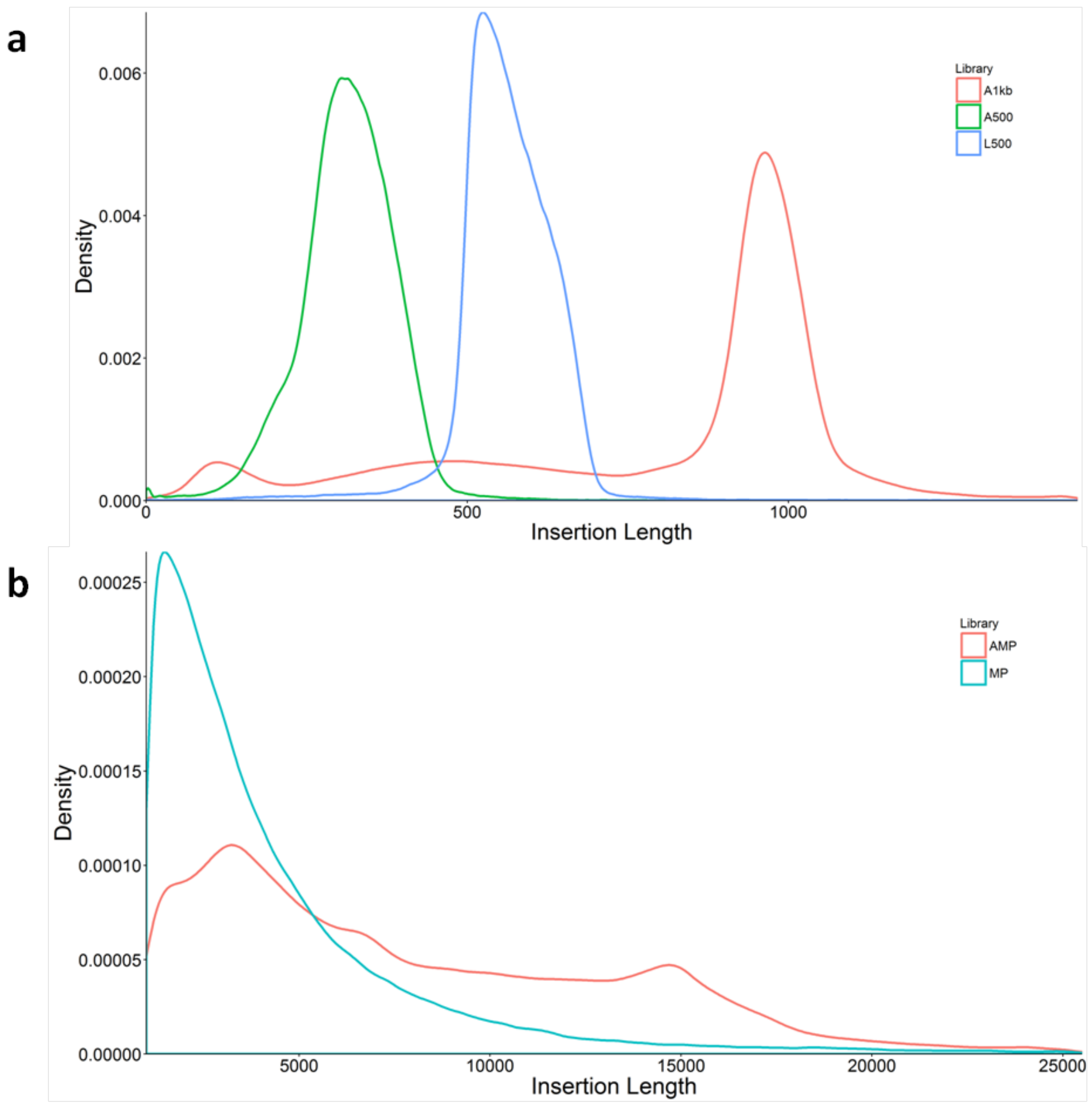


FIGURE 6.3: **The insert size distribution of sequenced paired-end libraries.** (a) Insert size distribution of paired-end libraries A500, L500, and A1kb. (b) Insert size distribution of mate-pair libraries MP and AMP.

in size (the highlighted scaffold in *Figure 6.4-a*). This figure also explains the different heterozygosity characteristics of *Bacillus pumilus* genome which isolates it from the scaffolds of *I. batatas*.

After excluding *Bacillus pumilus* genome, the largest scaffold of *I. batatas*, which harbours 54 genes, was 581kb (and contained 133 contigs). The N50 of all scaffolds was ∼60kb with a 5,649bp contig N50. The total number of scaffolds was 79,089 and their length varied between 312 and 3,723,026bp. There were 3,796 scaffolds longer than 60kb. Besides scaffolds, there were 991,314 contigs with a total length of ∼436Mb. Among these, 92,790 contigs were longer than 1kb harbouring a total of 175,679,534bp. These contigs mainly reflected heterozygosity of the hexaploid genome, since 97.35% of all contigs have been mapped back to scaffolds and one third of these contigs were found

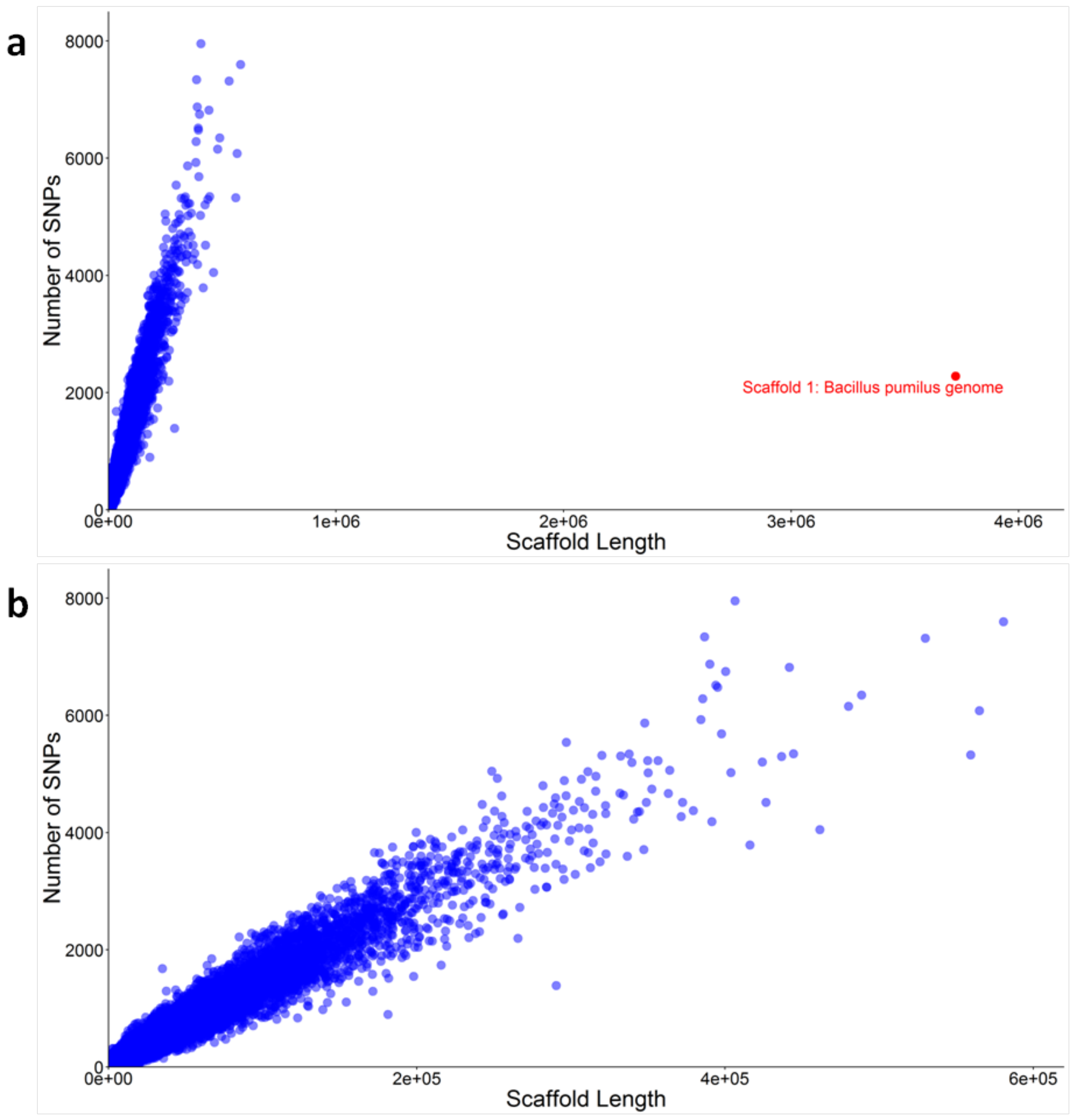


FIGURE 6.4: **Summary of variations in scaffolds in preliminary assembly.** (a) Number of variations and length of all scaffolds. An isolated dot represents the fully assembled genome of sweet potato endophyte, *Bacillus pumilus*. (b) High correlation (0.975) between "Number of variations" and "Scaffold length" after excluding the endophyte *Bacillus pumilus* genome.

to match at full length, despite many single nucleotide mismatches or small indels, as shown in *Figure 6.5*.

High heterozygosity prevents the assemblers to distinguish the sequences of different haplotypes of one genomic region; hence, they may assemble more than one sequence as contigs or scaffolds, which belong to one region. This causes the assembly of genomes that are larger than the estimated sizes. We designed three approaches to address the redundancy in the assembly. These approaches are based on applying the following modifications on the scaffolds and, then, exhaustive comparisons among modified sequences. The similarity between these sequences helps to decrease scaffolds' redundancies.

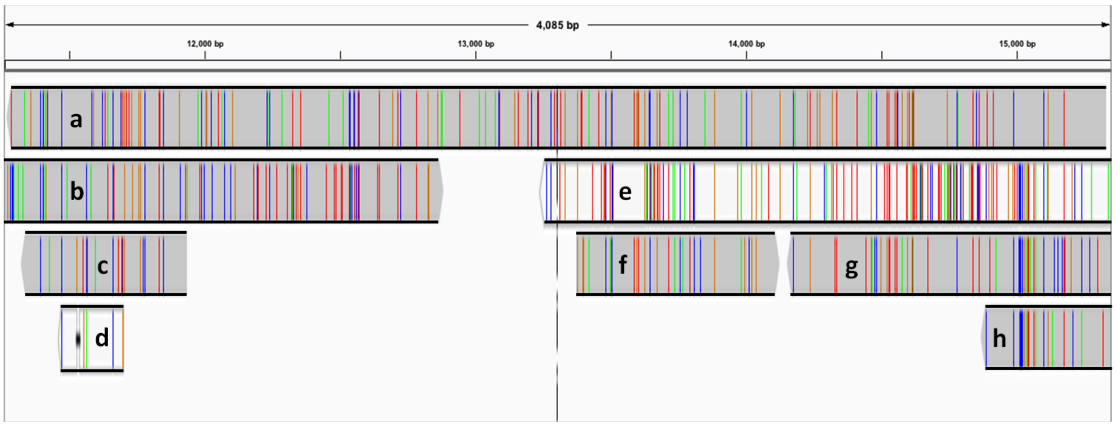


FIGURE 6.5: **A snapshot of mapping results of contigs against scaffolds.** The large number of single nucleotide mismatches indicates the single nucleotide polymorphism between homologous chromosomes in *Ipomoea batatas*. Cigar fields of these mapped scaffolds were listed as follows. (a) 4045M. (b) 33S44M1D136M1I10M6D149M15D21M1I1799M. (c) 579M. (d) 56M13D162M. (e) 15S2396M2I92M. (f) 733M. (g) 1260M. (h) 1014M1I162M1D136M. Among these, (b), (e), (g), and (h) are partially shown here.

1. **Replacing the variants by a sequence of wildcard nucleotides:** Knowing the length of the reference variants, we replace the variant sequences by sequences of wildcard nucleotides of the same length. Using this strategy, we are conserving the reference variant length and relaxing their sequence differences for the BLAT search[57].

2. **Removing the sequence variants from all scaffolds**: Having obtained the variant calling results, the reference alleles are removed from the scaffolds, and the modified scaffolds are considered as the input of the BLAT search. These variants are one of the reasons why the assemblers cannot distinguish that these scaffolds are actually from one region.

3. **Keep the reference variants in the scaffolds.**

Self-to-self BLAT was employed to check the similarity between the scaffolds and parallelized in computer farm. We then investigate the result based on sequence identity and the overlap length. When one scaffold was covered by another longer scaffold with more than $x\%$ sequence identity and more than $y\%$ sequence overlap, the shorter one was removed. *Table 6.2* details the results when $(x, y) = (80\%, 80\%)$ and $(x, y) = (85\%, 85\%)$.

As expected, the length and number of scaffolds obtained from method 1 were longer and more numerous than those from method 3 due to the relaxing condition achieved by using wildcard nucleotides. On the contrary, the results of method 2 show fewer scaffold maps; which is caused by the effect of the size and the frequency of indels in *I. batatas* genome (see *Figure 5.1-upper right* for the statistics of indel frequency).

	Method 1		Method 2		Method 3	
(Identity, Overlap)	Sum	#Scfs	Sum	#Scfs	Sum	#Scfs
(85%, 85%)	43M	17970	52M	22848	32M	14011
(80%, 80%)	59M	23009	120M	51075	47M	19106

Table 6.2: The result of self-to-self BLAT on all scaffolds by applying three strategies and two thresholds. This table summarizes the number of and the total length of removed scaffolds. *Overlap* shows the length of the overlapping sequence between scaffolds, and *Identity* indicates how similar the overlapping sequences are.

As the estimated size of the genome measured by C-value[58] was between 600Mb and 800Mb and the initial assembly size was ∼870Mb, we adopted the first method and used (85%,85%) thresholds. Prior to removing the candidate scaffolds, several long candidates were manually checked via Circos visualization[59].

After these procedures, there were 61,118 remaining scaffolds, of total length 822,598,598bp, with 64,561bp N50. Then these scaffolds were connected using 20kb mate-pair library by Platanus[60] ∼142kb. The total number of scaffolds was 57,051 and their length varied between 392 and 1,152,062bp. We call this version the "preliminary assembly".

6.4 Haplotype-Improved Assembly

In contrast with genome assembly, which relies on the similarity between the sequence reads, the haplotyping process pays more attention to DNA sequence differences among homologous chromosomes. The integration of genome assembly methods and haplotyping offers us a panoramic view of all the homologous chromosomes. Haplotyping itself, however, poses its own challenges. For example, the haplotyping of the human individual genome relies mainly on fosmid-based sequencing which is costly and time consuming[34, 61]. Since the distance of the adjacent variants between paternal and maternal chromosomes in humans is normally in the kilobase range[62], it is beyond the capacity of current cost-effective sequencing platforms to cover at least two variant positions in most cases. Nevertheless, human haplotyping studies have already indicated that genome assembly and accurate haplotyping are tightly linked[61]. Unfortunately, the computational phasing problem in polyploidy is considerably harder than for a diploid organism because in the polyploid case one can not make inferences about the "other" haplotype once one has seen the first.

Here, we report the integration of *de novo* assembly and haplotyping methods to improve the genome assembly. We recruit ***Ranbow*** haplotyper in an iterative manner to update haplotypes based on the assembly, and these haplotypes are then used for post-scaffolding. Using this pipeline, we observed remarkable improvements in terms of longer stretches of scaffolds, more phased regions, and longer haplotype lengths (see *Section 6.4.5*).

6.4.1 Variant calling

All the Illumina raw reads were mapped back to all scaffolds of the preliminary assembly (*Table 6.1*). After removing the PCR duplicates, there were 1,725,677,696 mapped reads in the final BAM file for variant calling using freebayes[63]. In total, there were 14,342,083 variations, consisting mainly of single nucleotide polymorphisms but also indels, across the assembly. Most of the variant positions harboured two possible alleles (*Figure 6.4.1-a*). We observed, on average, one polymorphic site every ∼58bp and a median distance of 20bp between polymorphic sites. The distance distribution peaked at 6bp and only 7% of observed distances are longer than 150bp (*Figure 6.6-b*). These findings confirmed our earlier conclusion that the *I. batatas* genome is very heterozygous and formed the basis for phasing haplotypes using 100–150bp Illumina reads. A high correlation (r=0.975) was found between number of variations and scaffold length (*Figure 6.4*), which increases our chances for phasing the scaffolds.

6.4.2 Haplotype phasing

We used ***Ranbow*** algorithm for reconstructing the haplotypes (see *Chapter 4).* The assembled haplotypes could be further extended by connecting paired-end reads. Part of the paired-end reads map to haplotypes within one scaffold, while other paired-end reads connect haplotypes from different scaffolds, due to better matching to the phased sequence. These connections are used for the Haplotype-Improved assembly. The assembled haplotypes length distribution is shown in *Figure 6.7.* The length distribution is shown in nucleotide and coded allele spaces, respectively. Moreover, we removed the gaps from the haplotypes to show the number of SmPs that are connected through the haplotypes.

6.4.3 Validation of haplotypes

To evaluate the haplotyping accuracy, we used a set of 454 reads(*Table 6.1-A454*) that had been produced earlier but were not used for assembly or phasing. Each 454 read can be considered as a short DNA fragment from one chromosome, except for some chimeric reads in rare cases. The reads are on the order of 1000bp long(*Figure 5.3)* and can thus serve to identify errors in the haplotype reconstruction. A large fraction of these

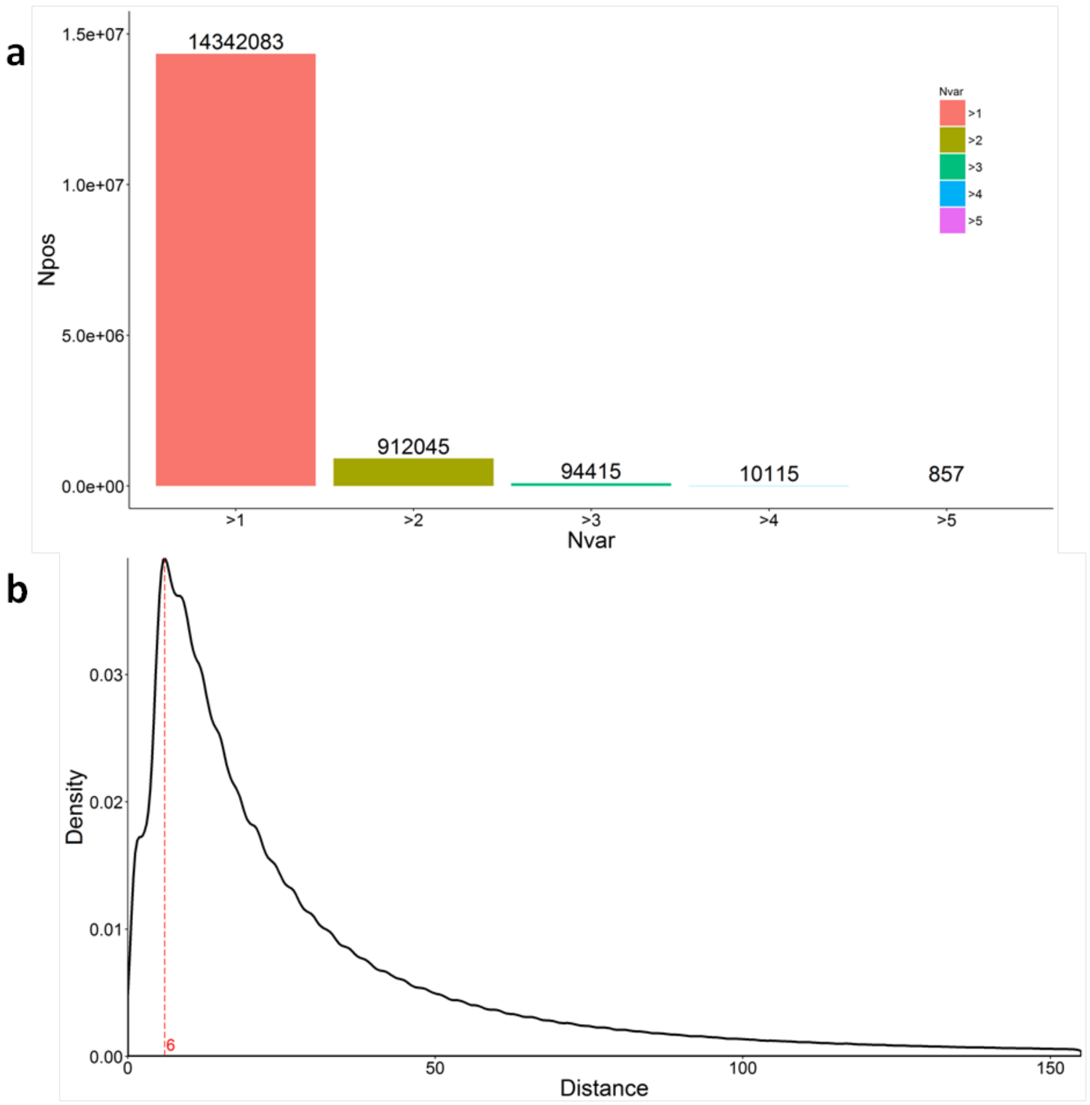


FIGURE 6.6: **Summary of variations.** (a) The total numbers of variation with different numbers of variant alleles. On average 831,919,670/14,342,083 = 58 bp with one variant, which means short reads (100bp and 150bp) are informative for haplotyping. Npos, Number of positions; Nvar, Number of Variant. (b) Adjacent variation distance distribution peaked at 6 bp (red dashed line). Only 7% observed distances are longer than 150bp.

454 reads displays haplotypes that have been correctly reconstructed by our short-read based methodology.

Roche 454-trimmed reads were mapped against genome assembly (please refer to *Section 5.2.2* and *Figure 5.3* for more details on the trimming procedure). Only the polymorphic sites indicated by variant calling were extracted and their overlaps with haplotypes were evaluated. The 'match' and 'mismatch' sites of each overlap, *i.e.*, the number of coinciding polymorphic sites between haplotype and 454 read, and the number of different polymorphic sites in the overlap respectively, were recorded for evaluating the haplotypes. More than 60% of overlaps between haplotypes and 454 reads are identical

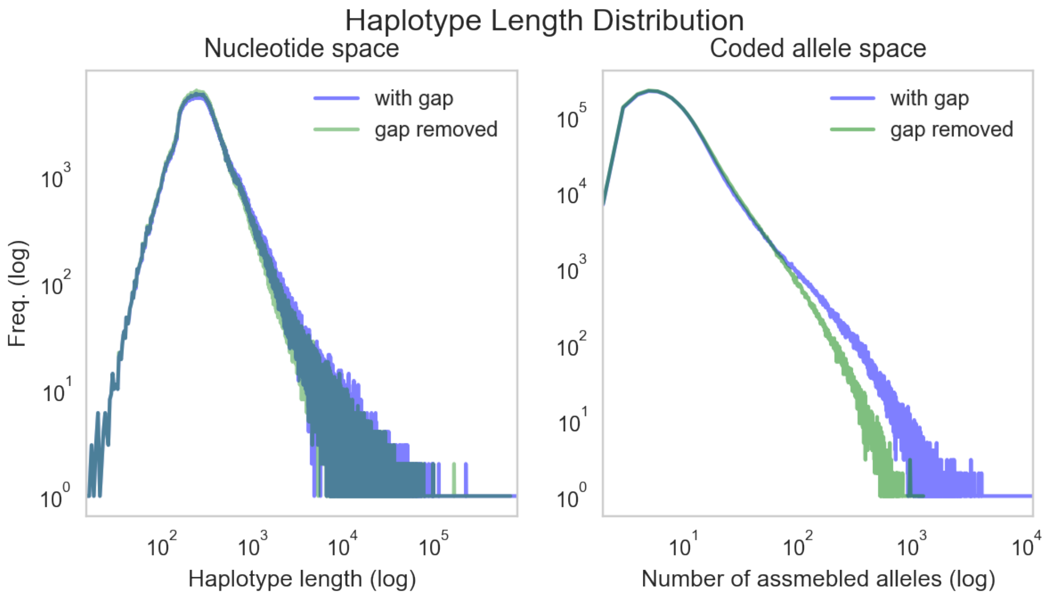


FIGURE 6.7: **Haplotype length distribution in nucleotide and coded allele levels.** These plots show the distribution of haplotype length as well as the number of gaps in them. In the coded allele space plot (the right one), we can see that both trends are covering each other when the number of SmPs is less than ∼100 alleles, and then they diverge as the length of haplotypes increase so that the maximum number of SmPs for normal haplotype is 11504 while it reduces to 1261 alleles by removing the gaps. This indicates the effect of paired-end and mate-pair reads which connect distant haplotypes. The divergence in nucleotide space is not as pronounced in coded allele space due to the availability of interpolymorphic regions.

at variant loci. The longest reconstructed haplotype contained 92 polymorphic sites without any mismatch indicated by 454 reads. There may be many longer perfectly reconstructed haplotypes that remain undetected because of the limited 454 read length.

6.4.4 Haplo-scaffolding strategy

Higher heterozygosity and ploidy in a genome complicates the scaffolding step of the *de novo* assembly pipeline. This, in part, is due to the assembly of erroneous reference which accumulates the most frequent alleles in polymorphic sites as the reference alleles. This results in a reference which differs from the true haplotypes. The problem arises when the reads are sequenced from the true haplotypes but are mapped to consensus reference. The higher the heterozygosity, the more the divergence between the true haplotypes and the assembled reference. In the scaffolding step, the reads are getting mapped to the reference sequence in order to find connections among scaffolds. Since higher heterozygosity increases the chance of observing more errors in the reference to which the reads are mapping, it influences the mapper efficiency. More reads are rejected to be mapped to the reference (*Figure 6.9(left scaffold)* shows an example of an inaccurate reference in a region with four polymorphic sites).

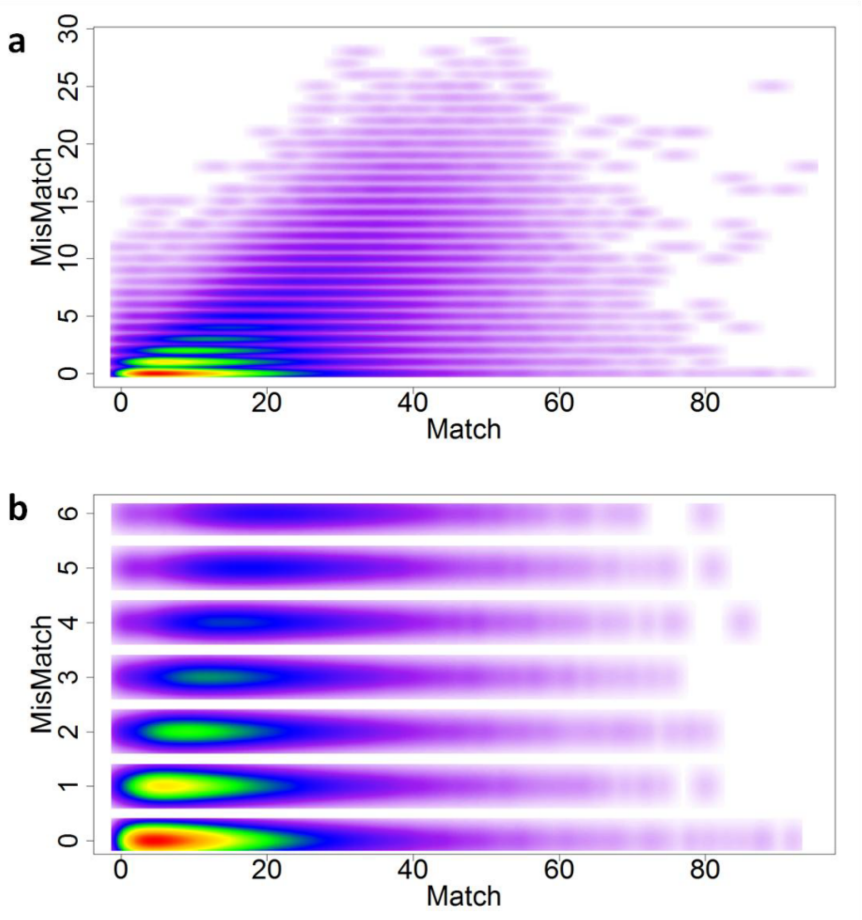


FIGURE 6.8: **Haplotype evaluation by 454 reads.** x axis: 'Match' is the number of coinciding polymorphic sites between haplotype and 454 read. y axis: 'Mismatch' is the number of different polymorphic sites in the overlap. Color indicates frequency of the respective (haplotype, 454 read) pairs, ranging from red to purple (on an exponential scale). (a) Evaluation of haplotypes. (b) Less than 6 mismatch part (97.25% of total number of overlaps). There were 64.78% of overlaps between haplotypes and 454 reads are identical at variant loci (y = 0). Even with strict mismatch threshold, a large fraction of 454 reads are supporting haplotypes reconstructed by short reads.

In order to solve the mentioned problem, we proposed a strategy called *haplo-scaffolding*. Firstly, we phase the genome into haplotypes which are packed in blocks. Each block is a fully phased region containing up to P haplotypes. Then, the paired-end reads are mapped to the haplotypes, and perfect maps are selected as the candidates for scaffolding. The position of candidates are then calculated according to the position of haplotypes in the scaffold and the read in the haplotypes. These candidates are considered as new connections between different scaffolds. *Figure 6.9* depicts an example of the effect of haplo-scaffolding strategy for two scaffolds. In this example, we set the mapping threshold to two base pairs. Both of the reads in scaffold I are rejected by mappers. After mapping reads to the haplotypes and extract the perfect maps, the connections between two scaffolds are obtained. These connections are then used for the scaffolding step.

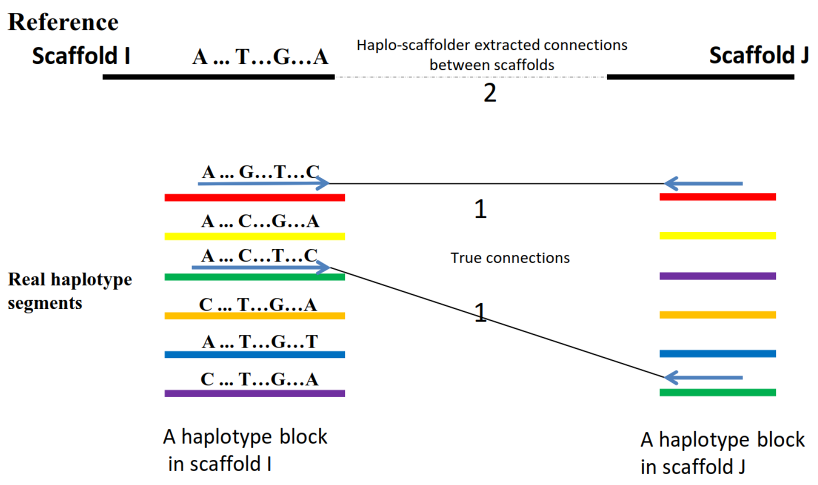


FIGURE 6.9: **The effect of haplo-scaffolder.** The reference accumulates the haplotypes' most frequent alleles. In scaffold I, it is shown that the reference sequence may not be similar to any of the haplotypes. Two reads, which are rejected from mapping due to the mismatch threshold, are now being mapped to the haplotypes (red and green haplotypes). The other reads of the pairs are mapped to scaffold J. These two newly extracted connection between scaffolds I and J could be used for haplo-scaffolding.

6.4.5 Haplotype-Improved assembly for *I. batatas* (HI-assembly)

All the Illumina raw reads were mapped against haplotype sequences generated by the phasing step. Only perfectly matched paired-end reads were considered as haplotype connections. The interscaffold and intrascaffold connections were separated for haplotype-based scaffolding and haplotype elongation, respectively. The interscaffold connections are separated according to insert sizes. It is worth noting that we faced the following technical issues by looking at the insert length distribution obtained from mapping AMP and MP libraries, 20kb and gel-free mate-pair reads respectively, to our assembly (see *Table 6.1*).

- We found two peaks for MP library (see *Figure 6.10* for the insert length distribution). To address this issue we extracted the reads mapped to the small scaffolds (<1000bp). It was expected that the first peak is constructed from the reads mapped to the substantial short scaffolds. By removing the reads mapped to the short scaffolds, the first peak disappeared. *Figure 6.11* indicates the proper peak for the MP library. Regarding the long tail of this library, we expected to see a wide insert length distribution since it was sequenced without selecting a size for the insert library (non-selected size insert library).

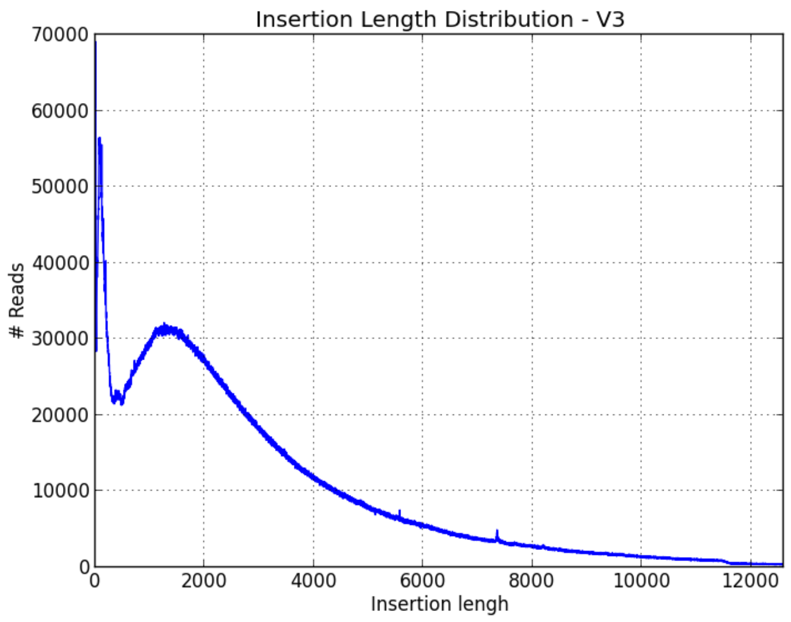


Figure 6.10: **Insert length distribution for MP library.** This library contains two peaks, one very short and one at 1.7kb, and a wide range of inserts up to ~12kb

- We couldn't find a clear peak of insert sizes for AMP library. In contrast to MP library, the insert length distribution for AMP was not anticipated to be in such a wide range without a clear peak around 20kb (*Figure 6.3-b(orange)*). It is worthwhile noting that AMP library was mapped to a closely related organism (*Ipomoea trifida*) as well to see if this problem was a result of mapping such long inserts to small scaffolds in our assembly or not. *Figure 6.12* illustrates that this trend is almost the same in both assemblies.

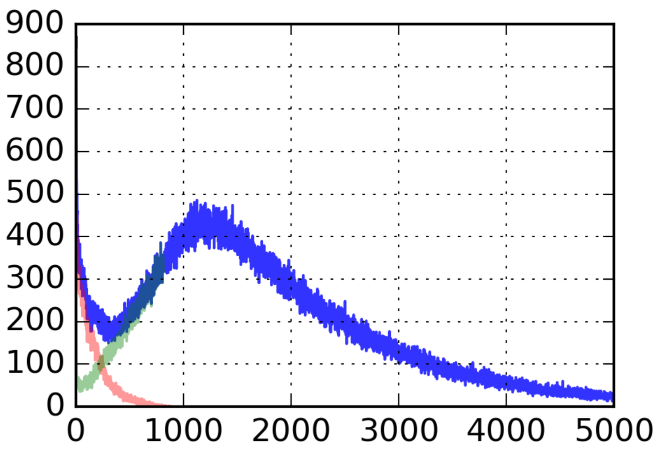


Figure 6.11: **The insert length distribution of MP library.** The insert sizes on short contigs (< 1000) are shown in red which explains the first peak on *Figure 6.10-a*. By removing the inserts mapped to small scaffolds the insert size becomes one peak distribution.

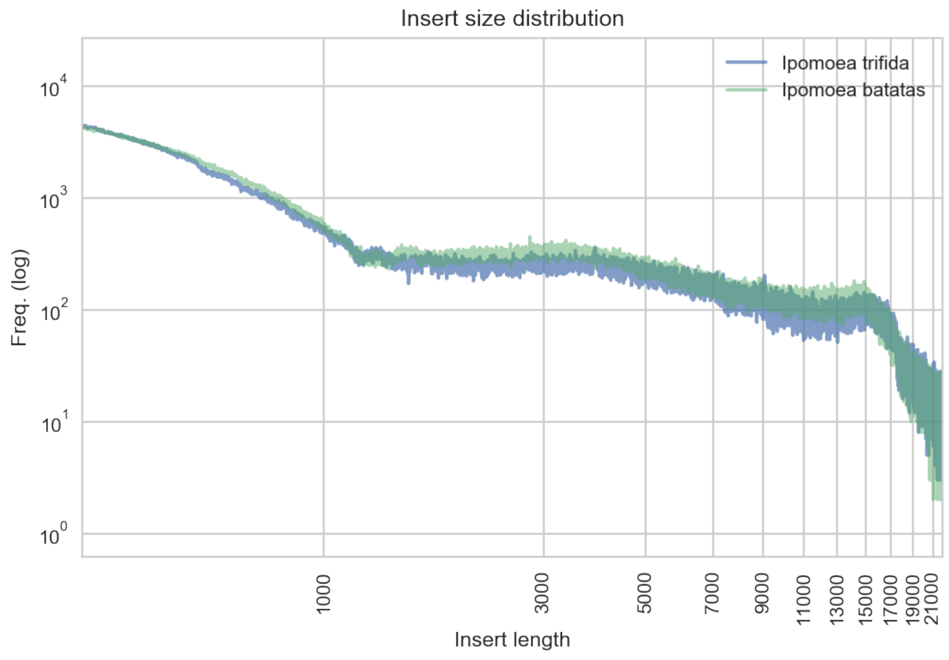


FIGURE 6.12: **Insert length distribution for AMP library**. One pronounced peak for very short inserts, two minor peaks at 3kb and 15kb.

We designed an approach for haplo-scaffolding considering mentioned technical issues. In this method, we extract the useful connections which are inferred from the assembled haplotypes. These connections are then used to assemble contigs and scaffolds into longer scaffolds by a scaffolding tool. We used SSPACE[64] scaffolder for this purpose.

Therefore, firstly, the connections from short insert size libraries, namely A500, L500, and A1kb, were applied. Among 994,434 connections obtained from the haplo-scaffolding step, 222,709 were mapped to the border of the scaffolds and used for scaffolding. These border regions are defined according to the insert size of sequence libraries. We set the minimum number of interscaffold connections to 5, the recommended value, which resulted in connecting ∼3500 scaffolds and 0.8% improvement of the N50 length. Afterwards, the connections obtained from MP and AMP libraries are selected according to predefined boundary sizes, *i.e.*, 1.5kb, 3kb, 6kb, 9kb, 12kb, and 15kb. These boundaries filter out the connections that are not placed in the scaffold borders. *Figure 6.13(lower left)* shows the proportion of reads which satisfy these conditions. Having the satisfying connections, the scaffolder was applied iteratively. At each step, a group of connections from one sequence library was fed to the scaffolder. The resulting scaffolds were considered as the input of the next step.

The major drawback of this approach is that it squeezes the gap sizes and results in smaller scaffolds due to combining both small and long connections in earlier steps, such as 1.5kb and 3kb inserts; and consequently, it has negative effects on the gap size prediction. Nevertheless, the Haplo-scaffolding resulted in 35,919 scaffolds (the input number of scaffolds was 57.051) varied between 392 and 1,335,955bp with the N50 of ∼201kb, 40% improvement.

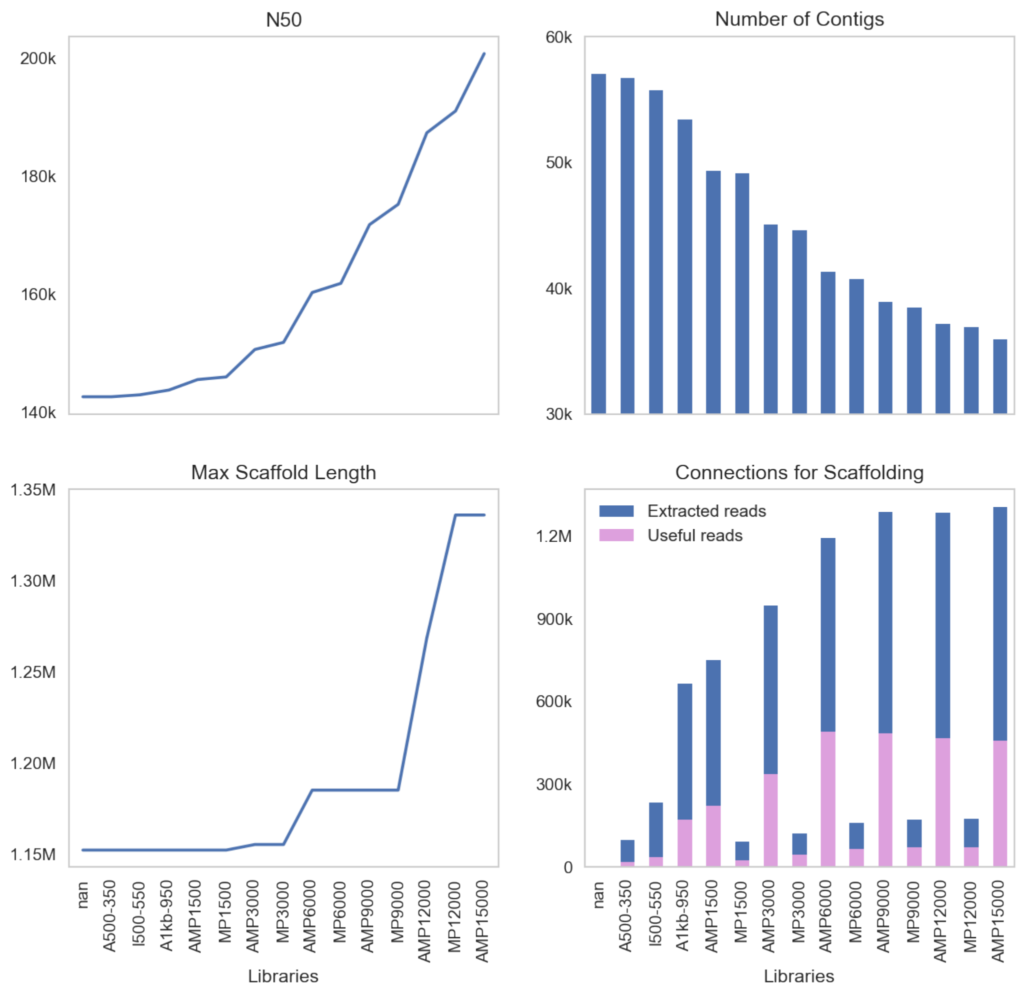


FIGURE 6.13: **Haplo-scaffolding results.** Upper left plot shows the improvement of N50 by applying different libraries. Lower left plot indicates the maximum scaffold length on each assembly. Upper right plot presents the histogram of number of scaffolds after applying each library. Lower right plot shows the number of extracted connection through haplo-scaffolders and the number of connections that are useful for haplo-scaffolding.

6.5 Conclusion

In this chapter, we explained the *de novo* assembly pipeline of the sweet potato genome. As the main part of this pipeline, we mentioned an application of haplotyping. We employed a haplotyping tool, ***Ranbow***, and designed a new haplotype aware scaffolder called haplo-scaffolders. Haplo-scaffolder uses the assembled haplotypes in order to rescue a set of potential connections which were hidden due to the differences of true haplotypes and the reference sequence. These connections are obtained by mapping the reads into haplotypes and transforming the connection information to the scaffold level. This process can be repeated as the set of scaffolds are updated after each round of haplo-scaffolding. We showed how the N50, maximum scaffold length improved by using this strategy for sweet potato genomes. The effect of haplo-scaffolding in the first iterations is high but after a number of iterations it reaches a plateau.

Summary

In this thesis, we focus on the problem of reconstructing haplotypes for polyploid genomes and the utilization of called haplotypes in *de novo* assembly of these genomes. We approach this topic exploring short read sequence data of the highly heterozygous hexaploid sweet potato genome.

First, we investigate the role of heterozygosity and ploidy number in reconstructing haplotypes with short reads. In short, higher heterozygosity provides higher number of useful reads for reconstructing haplotypes while being polyploid introduces a challenge in assembling reads into longer sequences; we called it the problem of 'Ambiguity of Merging fragments'. However, we address this problem and show that reads can be assembled into haplotypes with high accuracy using short reads. To this end, we propose a new algorithm, called ***Ranbow***, and evaluate its performance on real and simulated datasets from tetraploid *Capsella bursa-pastoris* (Shepherd's Purse), and hexaploid *Ipomoea batatas* (sweet potato) genomes. We are able to show that our method achieves higher accuracy and longer assembled haplotypes than the other methods.

Next, we present the *de novo* assembly pipeline of the sweet potato genome utilizing computed haplotypes for genome assembly improvement. This novel approach, called haplo-scaffolders, uses the assembled haplotypes in order to rescue a set of potential connections which were hidden due to the differences of true haplotypes and the reference sequence. These connections are obtained by mapping the reads into haplotypes and transforming the connection information to the reference level. This process can be repeated by updating the scaffold set to further improve the genome assembly. We show that this strategy improves substantially the N50 and maximum scaffold length of assembled sweet potato genome.

List of Figures

List of Tables

List of Algorithms

Abbreviations

DASE	Differential allele-specific expression
FPT	Fixed Parameter Tractable algorithm
GWAS	Genome Wide Association Study
HMM	Hidden Markov Model
HMW	High Molecular Weight
LD	Linkage Disequilibrium
MEC	Minimum Error Correction
MFR	Minimum Fragment Removal
NGS	Next Generation Sequencing
RNA	RiboNucleic Acid
SIH	Single Individual Haplotyping
SMRT	Single Molecule Real-Time
SNP	Single Nucleotide Polymorphism
SNV	Single Nucleotide Variation
WGS	Whole Genome Sequencing
wMEC	weighted Minimum Error Correction

Bibliography

[1] Rodger Staden. A strategy of DNA sequencing employing computer programs. *Nucleic acids research*, 6(7):2601–2610, 1979.

[2] Frederick Sanger, AR Coulson, T Friedmann, GM Air, BG Barrell, NL Brown, JC Fiddes, CA Hutchison III, PM Slocombe, and M Smith. The nucleotide sequence of bacteriophage φx174. *Journal of molecular biology*, 125(2):225–246, 1978.

[3] Robert D Fleischmann, Mark D Adams, Owen White, Rebecca A Clayton, Ewen F Kirkness, Anthony R Kerlavage, Carol J Bult, Jean-Francois Tomb, Brian A Dougherty, Joseph M Merrick, et al. Whole-genome random sequencing and assembly of haemophilus influenzae rd. *Science*, 269(5223):496–512, 1995.

[4] Mark D Adams, Susan E Celniker, Robert A Holt, Cheryl A Evans, Jeannine D Gocayne, Peter G Amanatides, Steven E Scherer, Peter W Li, Roger A Hoskins, Richard F Galle, et al. The genome sequence of drosophila melanogaster. *Science*, 287(5461):2185–2195, 2000.

[5] J Craig Venter, Mark D Adams, Eugene W Myers, Peter W Li, Richard J Mural, Granger G Sutton, Hamilton O Smith, Mark Yandell, Cheryl A Evans, Robert A Holt, et al. The sequence of the human genome. *Science*, 291(5507):1304–1351, 2001.

[6] R. C. Vrijenhoek. Polyploid hybrids: multiple origins of a treefrog species. *Current Biology*, 16(7):R245–247, Apr 2006.

[7] T. Yabe, X. Ge, and F. Pelegri. The zebrafish maternal-effect gene cellular atoll encodes the centriolar component sas-6 and defects in its paternal function promote whole genome duplication. *Dev. Biol.*, 312(1):44–60, Dec 2007.

[8] Sergej Nowoshilow, Siegfried Schloissnig, Ji-Feng Fei, Andreas Dahl, Andy WC Pang, Martin Pippel, Sylke Winkler, Alex R Hastie, George Young, Juliana G Roscito, et al. The axolotl genome and the evolution of key tissue formation regulators. *Nature*, 554(7690):50, 2018.

[9] Potato Genome Sequencing Consortium et al. Genome sequence and analysis of the tuber crop potato. *Nature*, 475(7355):189, 2011.

[10] Rachel Brenchley, Manuel Spannagl, Matthias Pfeifer, Gary LA Barker, Rosalinda D'Amore, Alexandra M Allen, Neil McKenzie, Melissa Kramer, Arnaud Kerhornou, Dan Bolser, et al. Analysis of the bread wheat genome using whole-genome shotgun sequencing. *Nature*, 491(7426):705, 2012.

[11] Jun Yang, M-Hossein Moeinzadeh, Heiner Kuhl, Johannes Helmuth, Peng Xiao, Stefan Haas, Guiling Liu, Jianli Zheng, Zhe Sun, Weijuan Fan, et al. Haplotype-resolved sweet potato genome traces back its hexaploidization history. *Nature Plants*, 3(9):696, 2017.

[12] Manuel Gonzalo Claros, Rocío Bautista, Darío Guerrero-Fernández, Hicham Benzerki, Pedro Seoane, and Noé Fernández-Pozo. Why assembling plant genome sequences is so challenging. *Biology*, 1(2):439–459, 2012.

[13] Yves Van de Peer, Eshchar Mizrachi, and Kathleen Marchal. The evolutionary significance of polyploidy. *Nature Reviews Genetics*, 18(7):411, 2017.

[14] Jared C Roach, Gustavo Glusman, Arian FA Smit, Chad D Huff, Robert Hubley, Paul T Shannon, Lee Rowen, Krishna P Pant, Nathan Goodman, Michael Bamshad, et al. Analysis of genetic inheritance in a family quartet by whole-genome sequencing. *Science*, 328(5978):636–639, 2010.

[15] Erez Lieberman-Aiden, Nynke L Van Berkum, Louise Williams, Maxim Imakaev, Tobias Ragoczy, Agnes Telling, Ido Amit, Bryan R Lajoie, Peter J Sabo, Michael O Dorschner, et al. Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *Science*, 326(5950):289–293, 2009.

[16] Robert R Graham, Chieko Kyogoku, Snaevar Sigurdsson, Irina A Vlasova, Leela RL Davies, Emily C Baechler, Robert M Plenge, Thearith Koeuth, Ward A Ortmann, Geoffrey Hom, et al. Three functional variants of ifn regulatory factor 5 (*irf5*)

define risk and protective haplotypes for human lupus. *Proceedings of the National Academy of Sciences*, 104(16):6758–6763, 2007.

[17] Robert R Graham, Chris Cotsapas, Leela Davies, Rachel Hackett, Christopher J Lessard, Joanlise M Leon, Noel P Burtt, Candace Guiducci, Melissa Parkin, Casey Gates, et al. Genetic variants near *TNFAIP3* on 6q23 are associated with systemic lupus erythematosus. *Nature genetics*, 40(9):1059, 2008.

[18] Connie M Drysdale, Dennis W McGraw, Catharine B Stack, J Claiborne Stephens, Richard S Judson, Krishnan Nandabalan, Kevin Arnold, Gualberto Ruano, and Stephen B Liggett. Complex promoter and coding region β2-adrenergic receptor haplotypes alter receptor expression and predict in vivo responsiveness. *Proceedings of the National Academy of Sciences*, 97(19):10483–10488, 2000.

[19] Stephan Schiffels and Richard Durbin. Inferring human population size and separation history from multiple genome sequences. *Nature genetics*, 46(8):919, 2014.

[20] Andrew G Clark. Inference of haplotypes from PCR-amplified samples of diploid populations. *Molecular biology and evolution*, 7(2):111–122, 1990.

[21] Sharon R Browning and Brian L Browning. Haplotype phasing: existing methods and new developments. *Nature Reviews Genetics*, 12(10):703, 2011.

[22] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

[23] Mark A Batzer and Prescott L Deininger. Alu repeats and human genomic diversity. *Nature reviews genetics*, 3(5):370, 2002.

[24] Todd J Treangen and Steven L Salzberg. Repetitive DNA and next-generation sequencing: computational challenges and solutions. *Nature Reviews Genetics*, 13 (1):36, 2012.

[25] Matthew Pendleton, Robert Sebra, Andy Wing Chun Pang, Ajay Ummat, Oscar Franzen, Tobias Rausch, Adrian M Stütz, William Stedman, Thomas Anantharaman, Alex Hastie, et al. Assembly and diploid architecture of an individual human genome via single-molecule technologies. *Nature methods*, 12(8):780, 2015.

[26] Chen-Shan Chin, Paul Peluso, Fritz J Sedlazeck, Maria Nattestad, Gregory T Concepcion, Alicia Clum, Christopher Dunn, Ronan O'Malley, Rosa Figueroa-Balderas,

Abraham Morales-Cruz, et al. Phased diploid genome assembly with single-molecule real-time sequencing. *Nature methods*, 13(12):1050, 2016.

[27] https://emea.illumina.com/systems/sequencing-platforms.html.

[28] Rudi Cilibrasi, Leo Van Iersel, Steven Kelk, and John Tromp. On the complexity of several haplotyping problems. In *International Workshop on Algorithms in Bioinformatics*, pages 128–139. Springer, 2005.

[29] Alessandro Panconesi and Mauro Sozio. Fast hare: A fast heuristic for single individual SNP haplotype reconstruction. In *International workshop on algorithms in bioinformatics*, pages 266–277. Springer, 2004.

[30] Eugene W Myers, Granger G Sutton, Art L Delcher, Ian M Dew, Dan P Fasulo, Michael J Flanigan, Saul A Kravitz, Clark M Mobarry, Knut HJ Reinert, Karin A Remington, et al. A whole-genome assembly of drosophila. *Science*, 287(5461): 2196–2204, 2000.

[31] Samuel Levy, Granger Sutton, Pauline C Ng, Lars Feuk, Aaron L Halpern, Brian P Walenz, Nelson Axelrod, Jiaqi Huang, Ewen F Kirkness, Gennady Denisov, et al. The diploid genome sequence of an individual human. *PLoS Biology*, 5(10):e254, 2007.

[32] Vikas Bansal and Vineet Bafna. Hapcut: an efficient and accurate algorithm for the haplotype assembly problem. *Bioinformatics*, 24(16):i153–i159, 2008.

[33] Harry R Lewis. Computers and intractability. a guide to the theory of np-completeness, 1983.

[34] Jorge Duitama, Gayle K. McEwen, Thomas Huebsch, Stefanie Palczewski, Sabrina Schulz, Kevin Verstrepen, Eun-Kyung Suk, and Margret R. Hoehe. Fosmid-based whole genome haplotyping of a hapmap trio child: evaluation of single individual haplotyping techniques. *Nucleic Acids Research*, 40(5):2041–2053, 2012. doi: 10.1093/nar/gkr1042. URL +http://dx.doi.org/10.1093/nar/gkr1042.

[35] URL http://slideplayer.com/slide/5071817/.

[36] Ryan Tewhey, Vikas Bansal, Ali Torkamani, Eric J Topol, and Nicholas J Schork. The importance of phase information for human genomics. *Nature Reviews Genetics*, 12(3):215, 2011.

[37] Wen-Yun Yang, Farhad Hormozdiari, Zhanyong Wang, Dan He, Bogdan Pasaniuc, and Eleazar Eskin. Leveraging reads that span multiple single nucleotide polymorphisms for haplotype inference from sequencing data. *Bioinformatics*, 29(18): 2245–2252, 2013.

[38] Na Li and Matthew Stephens. Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, 165 (4):2213–2233, 2003.

[39] Emily Berger, Deniz Yorukoglu, Jian Peng, and Bonnie Berger. Haptree: A novel bayesian framework for single individual polyplotyping using NGS data. *PLoS Computational Biology*, 10(3):e1003502, 2014.

[40] Emily Berger, Deniz Yorukoglu, and Bonnie Berger. Haptree-x: An integrative bayesian framework for haplotype reconstruction from transcriptome and genome sequencing data. In *International Conference on Research in Computational Molecular Biology*, pages 28–29. Springer, 2015.

[41] H Christina Fan, Jianbin Wang, Anastasia Potanina, and Stephen R Quake. Whole-genome molecular haplotyping of single cells. *Nature biotechnology*, 29(1):51, 2011.

[42] Grace XY Zheng, Billy T Lau, Michael Schnall-Levin, Mirna Jarosz, John M Bell, Christopher M Hindson, Sofia Kyriazopoulou-Panagiotopoulou, Donald A Masquelier, Landon Merrill, Jessica M Terry, et al. Haplotyping germline and cancer genomes with high-throughput linked-read sequencing. *Nature biotechnology*, 34 (3):303, 2016.

[43] Jacob O Kitzman. Haplotypes drop by drop. *Nature biotechnology*, 34(3):296, 2016.

[44] David Porubskỳ, Ashley D Sanders, Niek van Wietmarschen, Ester Falconer, Mark Hills, Diana CJ Spierings, Marianna R Bevova, Victor Guryev, and Peter M Lansdorp. Direct chromosome-length haplotyping by single-cell sequencing. *Genome research*, 26(11):1565–1574, 2016.

[45] Volodymyr Kuleshov. Probabilistic single-individual haplotyping. *Bioinformatics*, 30(17):i379–i385, 2014.

[46] Sergey Koren, Michael C Schatz, Brian P Walenz, Jeffrey Martin, Jason T Howard, Ganeshkumar Ganapathy, Zhong Wang, David A Rasko, W Richard McCombie,

Erich D Jarvis, et al. Hybrid error correction and *de novo* assembly of single-molecule sequencing reads. *Nature biotechnology*, 30(7):693, 2012.

[47] Derek Aguiar and Sorin Istrail. Haplotype assembly in polyploid genomes and identical by descent shared tracts. *Bioinformatics*, 29(13):i352–i360, 2013.

[48] Shreepriya Das and Haris Vikalo. Sdhap: haplotype assembly for diploids and polyploids via semi-definite programming. *BMC genomics*, 16(1):260, 2015.

[49] Minzhu Xie, Qiong Wu, Jianxin Wang, and Tao Jiang. H-PoP and H-PoPG: heuristic partitioning algorithms for single individual haplotyping of polyploids. *Bioinformatics*, 32(24):3735–3744, 2016.

[50] Artem S Kasianov, Anna V Klepikova, Ivan V Kulakovskiy, Evgeny S Gerasimov, Anna V Fedotova, Elizaveta G Besedina, Alexey S Kondrashov, Maria D Logacheva, and Aleksey A Penin. High quality genome assembly of *capsella bursa-pastoris* reveals asymmetry of regulatory elements at early stages of polyploid genome evolution. *The Plant Journal*, 2017.

[51] L. Janin. Enhanced artificial genome engine: next generation sequencing reads simulator. *GitHub repository: https://github.com/sequencing/EAGLE*, commit:a3215138846ca3bd969093214163ede015835a10, 2014.

[52] Xuesong Hu, Jianying Yuan, Yujian Shi, Jianliang Lu, Binghang Liu, Zhenyu Li, Yanxiang Chen, Desheng Mu, Hao Zhang, Nan Li, et al. pirs: Profile-based illumina pair-end reads simulator. *Bioinformatics*, 28(11):1533–1535, 2012.

[53] Kittipat Ukoskit and Paul G Thompson. Autopolyploidy versus allopolyploidy and low-density randomly amplified polymorphic DNA linkage maps of sweetpotato. *Journal of the American Society for Horticultural Science*, 122(6):822–828, 1997.

[54] Albert Kriegner, Jim Carlos Cervantes, Kornel Burg, Robert OM Mwanga, and Dapeng Zhang. A genetic linkage map of sweetpotato [*ipomoea batatas (l.) lam.*] based on AFLP markers. *Molecular Breeding*, 11(3):169–185, 2003.

[55] Yu Peng, Henry C. M. Leung, S. M. Yiu, and Francis Y. L. Chin. IDBA-UD: a *de novo* assembler for single-cell and metagenomic sequencing data with highly uneven depth. *Bioinformatics*, 28(11):1420–1428, 2012. doi: 10.1093/bioinformatics/bts174. URL +http://dx.doi.org/10.1093/bioinformatics/bts174.

[56] Marcel Margulies, Michael Egholm, William E Altman, Said Attiya, Joel S Bader, Lisa A Bemben, Jan Berka, Michael S Braverman, Yi-Ju Chen, Zhoutao Chen, et al. Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437 (7057):376, 2005.

[57] W James Kent. BLAT—the BLAST-like alignment tool. *Genome research*, 12(4): 656–664, 2002.

[58] Peggy Ozias-Akins and Robert L Jarret. Nuclear DNA content and ploidy levels in the genus *ipomoea*. *Journal of the American Society for Horticultural Science*, 119 (1):110–115, 1994.

[59] Martin I Krzywinski, Jacqueline E Schein, Inanc Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J Jones, and Marco A Marra. Circos: an information aesthetic for comparative genomics. *Genome research*, 2009.

[60] Rei Kajitani, Kouta Toshimoto, Hideki Noguchi, Atsushi Toyoda, Yoshitoshi Ogura, Miki Okuno, Mitsuru Yabana, Masayuki Harada, Eiji Nagayasu, Haruhiko Maruyama, et al. Efficient *de novo* assembly of highly heterozygous genomes from whole-genome shotgun short reads. *Genome research*, 24(8):1384–1395, 2014.

[61] Ruibang Luo, Binghang Liu, Yinlong Xie, Zhenyu Li, Weihua Huang, Jianying Yuan, Guangzhu He, Yanxiang Chen, Qi Pan, Yunjie Liu, Jingbo Tang, Gengxiong Wu, Hao Zhang, Yujian Shi, Yong Liu, Chang Yu, Bo Wang, Yao Lu, Changlei Han, David W Cheung, Siu-Ming Yiu, Shaoliang Peng, Zhu Xiaoqian, Guangming Liu, Xiangke Liao, Yingrui Li, Huanming Yang, Jian Wang, Tak-Wah Lam, and Jun Wang. Soapdenovo2: an empirically improved memory-efficient short-read *de novo* assembler. *GigaScience*, 1(1):1–6, 2012. doi: 10.1186/2047-217X-1-18. URL +http://dx.doi.org/10.1186/2047-217X-1-18.

[62] Hongzhi Cao, Honglong Wu, Ruibang Luo, Shujia Huang, Yuhui Sun, Xin Tong, Yinlong Xie, Binghang Liu, Hailong Yang, Hancheng Zheng, et al. *De novo* assembly of a haplotype-resolved human genome. *Nature biotechnology*, 33(6):617, 2015.

[63] E. Garrison and G. Marth. Haplotype-based variant detection from short-read sequencing. *Preprint at http://arxiv.org/abs/1207.3907v2*, 2012.

[64] Marten Boetzer, Christiaan V Henkel, Hans J Jansen, Derek Butler, and Walter Pirovano. Scaffolding pre-assembled contigs using SSPACE. *Bioinformatics*, 27(4): 578–579, 2010.

[65] Sudhir Kumar, Glen Stecher, Daniel Peterson, and Koichiro Tamura. MEGA-CC: computing core of molecular evolutionary genetics analysis program for automated and iterative data analysis. *Bioinformatics*, 28(20):2685–2686, 2012.

Appendix A

Ranbow user manual

Ranbow is a haplotype assembler for polyploid genomes. Initially, it has been developed for the haplotype assembly of the sweet potato genome, which is hexaploid and highly heterozygous. ***Ranbow*** can also be applied to other polyploid genomes. After one round of haplotype assembly, ***Ranbow*** utilizes the assembled haplotypes to correct called variants. Moreover, it can infer the organism evolutionary history for computed haplotypes. ***Ranbow*** has three main modes of function:

`ranbow hap:` for haplotyping

`ranbow eval:` for evaluating of the assemble haplotypes by gold standard (long) reads

`ranbow phylo:` for the phylogenetic analysis

Availability

You can download the source code and the toy example files here:

`https://www.molgen.mpg.de/ranbow`

The following video explains how you can simply run ***Ranbow*** on the toy example:

`https://youtu.be/2zZ5IfpuGZQ`

Dependencies

The code is implemented in python 2.7.13 and can be run from linux shell. To run ***Ranbow*** the following libraries and tools should be available:

python libraries

- pysam 0.10.0
- numpy 1.12.1

tools

- samtools 0.1.19-44428cd.

A.1 Data preparation

We generated a random list of scaffolds from the sweet potato genome in order to make a data input as a toy example. Then, the *fasta*, *bam* and *vcf* files were extracted. Data files for selected scaffolds are named as 'sel.bam', 'sel.vcf', and 'sel.fasta'. These files are generated by 'toy_data.sh' which is available in toy folder as well.

It worth mentioning that ***Ranbow*** can be applied on the whole or selected region of the scaffolds. This feature helps for better parallelization as well as phasing targeted interval in the genome.

A.2 *Ranbow* for haplotype assembly

`Ranbow hap` has four modes of sub-function. For simplicity we refer to them as different modes, namely `index`, `hap`, `collect`, and `modVCF`. The mode of choice is declared by `-mode` parameter in command line.

```
python2_7_13 ranbow.py hap -mode index
python2_7_13 ranbow.py hap -mode hap python2_7_13 ranbow.py hap -mode collect
python2_7_13 ranbow.py hap -mode modVCF
```

The `index` mode generates the index files in order to have random access to chromosomes and scaffolds in case of parallelization. After running ***Ranbow*** in `index` and `hap` mode, ***Ranbow*** should be applied on `collect` mode. The `collect` mode collects the haplotypes assembled with different processors to generate the final *fasta* file, *bam* file (aligned haplotypes) and also *hap* formatted file. The *bam* file is already sorted and indexed and is ready to be loaded in IGV browser for downstream analysis. The *hap* formatted file includes haplotype information such as name, start position, haplotype sequence, number of error correction, haplotype quality, and supporting fragments. These fields are explained in more details as follows.

Haplotype name: The name includes scaffold/chromosome name, and an index number at the end of the name. For example if the ploidy is six, the following haplotypes belong to the haplotype block indexed by 4 in chromosome 1: chr1_hap4_0, chr1_hap4_1, chr1_hap4_2, chr1_hap4_3, chr1_hap4_4, and chr1_hap4_5.

Haplotype sequence: sequence of alleles in which the alleles are coded to numbers in [0, ploidy) interval. For sweet potato they are coded with '0' to '5'.

Number of error correction: Each haplotype has a number of supporting fragments. This field represents the number of mismatches among these fragments.

Quality of haplotype: The supporting fragment coverage of each position in the haplotype. Length of quality is equal to the length of haplotype. **Supporting fragments:** A full list of supporting fragments. To run ***Ranbow***, the parameters in the following table need to be adjusted. Some of these parameters have to be passed from the command line (mentioned as following), while the others could also be passed via a parameter file. The compulsory parameters are as follows: Here is the content of a parameter file used in this toy example:

```
-ploidy 6
-noProcessor 4
-bamFile path to folder∼>RANBOW/toy/sel.bam
-refFile path to folder∼>RANBOW/toy/sel.fasta
-vcfFile path to folder∼>RANBOW/toy/sel.vcf
-selectedScf path to folder∼>RANBOW/toy/scaffolds.list
-outputFolderBase path to folder∼>/RANBOW/toy/result
```

For simplicity, data files are stored in one folder (named 'RANBOW/toy') and the parameters are adjusted in 'hap.params' file. The parameter file is passed to ***Ranbow*** through the `-par` argument. `python2_7_13 ranbow.py hap -par RANBOW/toy/hap.params` So here is the list of files in the folder:

```
RANBOW/toy > ls -lh
total 30M
```

```
-rw-r----- 1 moeinzad moeinzad 426 Mar 31 10:33 hap.params
-rw-r----- 1 moeinzad moeinzad 641 Mar 30 09:51 scaffolds.list
-rw-r----- 1 moeinzad moeinzad 29M Mar 30 11:21 sel.bam
-rw-r----- 1 moeinzad moeinzad 133K Mar 30 11:21 sel.fasta
-rw-r----- 1 moeinzad moeinzad 1.1M Mar 30 11:21 sel.vcf
-rw-r----- 1 moeinzad moeinzad 405 Mar 30 11:18 toy_data.sh
```

A.2.1 Indexing input files (mode: index)

It generates index files for *bam*, *vcf*, and *fasta* if not exist:

```
python2_7_13 ranbow.py hap -mode index -par RANBOW/toy/hap.params
```

Then the index files are generated and listed as follows:

```
RANBOW/toy > ls -lht
total 30M
drwxr-x--- 2 moeinzad moeinzad 10 Mar 31 10:58 result
-rw-r----- 1 moeinzad moeinzad 844 Mar 31 10:58 sel.vcf.index
-rw-r----- 1 moeinzad moeinzad 225K Mar 31 10:58 sel.bam.bai
-rw-r----- 1 moeinzad moeinzad 1.2K Mar 31 10:58 sel.fasta.fai
-rw-r----- 1 moeinzad moeinzad 426 Mar 31 10:33 hap.params
-rw-r----- 1 moeinzad moeinzad 29M Mar 30 11:21 sel.bam
-rw-r----- 1 moeinzad moeinzad 1.1M Mar 30 11:21 sel.vcf
-rw-r----- 1 moeinzad moeinzad 133K Mar 30 11:21 sel.fasta
-rw-r----- 1 moeinzad moeinzad 405 Mar 30 11:18 toy_data.sh
-rw-r----- 1 moeinzad moeinzad 641 Mar 30 09:51 scaffolds.list
```

A.2.2 Run *Ranbow* on computer farm

The `-noProcessor` parameter should be adjusted according to the number of available processors. For example, if the code is running on a cluster with 200 cores, `-noProcessor` can be set to 200. Then, 200 independent jobs will be executed with different set of chromosomes, chromosome parts, or scaffolds. These 200 jobs are independent and can be run on different machines. For the sake of simplicity in the toy example, we run the code utilizing three cores.

The `-processorIndex` is a compulsory parameter in command line if the number of processors is more than one (`-noProcessor` ≥ 2). Otherwise `-noProcessor` is set to one and `-processorIndex` is set to zero by default, meaning that the output is generated with one processor and the result is going to be generated in a folder named '0'. Here, the standard output for the first processor (`-processorIndex` = 0 and `-noProcessor` = 3) is shown as follows.

```
python2_7_13 ranbow.py hap -mode hap -par hap.params -processorIndex 0 scaffold4183|si
readBAM 0:00:12.360045 Ranbow_single 0:00:51.672871 single2file 0:00:00.055484
scaffold9406|size6143 readBAM 0:00:01.335690 Ranbow_single 0:00:00.000836
single2file 0:00:00.000018 scaffold13332|size3918 readBAM 0:00:03.531090 Ranbow_single
0:00:00.162100 single2file 0:00:00.002223 scaffold16724|size3385 readBAM 0:00:01.68417
Ranbow_single 0:00:00.201964 single2file 0:00:00.003162
```

The running time for each part of haplotyping is reported in the standard output. The haplotyping result will be appear in the subfolder of `-outputFolderBase` named as `-processorIndex`. For example, since we generated haplotypes for `-processorIndex 0`, the only generated folder is called '0'

```
RANBOW/toy/result > ls
0
```

This folder contains three files as follows:

```
RANBOW/toy/result/0 > ls -lh
total 964K
-rw-r----- 1 moeinzad moeinzad 49K Mar 31 11:01 ranbow.single.hap
-rw-r----- 1 moeinzad moeinzad 113K Mar 31 11:01 ranbow.single.hap.fasta
-rw-r----- 1 moeinzad moeinzad 234K Mar 31 11:01 ranbow.single.hap.bam
```

The *hap* format shows the connectivity between alleles in haplotype segments, the quality of alleles in assembled haplotype (the number of supporting reads for the allele), and in general all information regarding the assembled haplotype in coded allele space. In coded allele space means the shared sequence between alleles are ignored and the alleles are decoded to numbers. The *fasta* and the *bam* files for the aligned haplotypes are generated in this folder as well.

To run the code in parallel on one machine the following command can be executed:

```
for i in {0..2}
do
python2_7_13 ranbow.py hap -mode hap -par hap.params -processorIndex $i > $i.log
&
done
```

The standard outputs for every processor is collected in '0.log', '1.log', and '2.log' files.

It is also possible to run ***Ranbow*** partly in one machine and partly in the other. Following our toy example, set 0 (`-processorIndex = 0`) and set 1 (`-processorIndex = 1`) are executed in machine A and set 2 (`-processorIndex = 2`) is executed in machine B.

```
Machine A:
for i in {0..1}
do
python2_7_13 ranbow.py hap -mode hap -par hap.params -processorIndex i >i.log
&
done Machine B:
for i in {2..2}
do
python2_7_13 ranbow.py hap -mode hap -par hap.params -processorIndex i >i.log
&
done
```

The three generated folder are listed as follows:

```
 RANBOW/toy/result > ls -lh
total 12K
drwxr-x--- 2 moeinzad moeinzad 4.0K Mar 31 11:00 0
drwxr-x--- 2 moeinzad moeinzad 4.0K Mar 31 11:03 1
drwxr-x--- 2 moeinzad moeinzad 4.0K Mar 31 11:03 2
```

A.2.3 Collecting data from (mode: collect)

When all jobs get finished, `-mode collect` can be executed to collect the haplotypes from different machines:

```
 python2_7_13 ranbow.py hap -mode collect -par hap.params
number of folders:  3
[samopen] SAM header is present:  28 sequences.
[samopen] SAM header is present:  28 sequences.
```

Then the generated files are as follows:

```
 RANBOW/toy/result > ls -lht
total 1.2M
-rw-r----- 1 moeinzad moeinzad 359K Mar 31 11:05 ranbow.single.hap.fasta
-rw-r----- 1 moeinzad moeinzad 132K Mar 31 11:05 ranbow.single.hap
-rw-r----- 1 moeinzad moeinzad 2.3K Mar 31 11:05 ranbow.single.hap.bam.bai
-rw-r----- 1 moeinzad moeinzad 48K Mar 31 11:05 ranbow.single.hap.bam
drwxr-x--- 2 moeinzad moeinzad 4.0K Mar 31 11:03 2
drwxr-x--- 2 moeinzad moeinzad 4.0K Mar 31 11:03 1
drwxr-x--- 2 moeinzad moeinzad 4.0K Mar 31 11:00 0
```

For this toy example the generated ‘single.hap.bam’ file is ready to be loaded for IGV viewer for further analysis.

A.2.4 Revising sequence variants (mode: modVCF)

The following command modifies and corrects the variants with the aid of assembled haplotypes.

```
hap/RANBOW > python ranbow.py hap -mode modVCF -par params.txt
```

modified vcf file:

```
/hap/RANBOW/toy/sel.mod.vcf
```

log of modified SNPs:

```
/hap/RANBOW/toy/result/ranbow.modVCF.log
Modified SNPs:  853 Deleted SNPs:  49
```

The detailed information of the modification can be found in:

```
/hap/RANBOW/toy/result/ranbow.modVCF.log
```

A.2.5 Evaluation of the results

To evaluate the haplotyping accuracy, we recruit the Roche 454 trimmed reads mapped to the assembly. This file or other gold standard mapped read files has to be passed with `-bamFileEval` parameter. `ranbow eval` can also be executed in parallel. For obtaining the evaluation result the following steps needs to be done.

```
python2_7_13 ranbow.py eval -par hap.params -mode index
```

`mode run` for parallelization:

```
python2_7_13 ranbow.py eval -par hap.params -mode run -processorIndex i
```

`mode collect` for collecting data:

```
python2_7_13 ranbow.py eval -par hap.params -mode collect
```

After running the above commands the following files will be generated:

```
 toy/result > ls -lt eval/
total 28
-rw-r----- 1 moeinzad moeinzad 4561 Apr 16 18:33 result.sing
drwxr-x--- 2 moeinzad moeinzad 66 Apr 16 18:33 0
drwxr-x--- 2 moeinzad moeinzad 66 Apr 16 18:33 1
drwxr-x--- 2 moeinzad moeinzad 66 Apr 16 18:33 2
drwxr-x--- 2 moeinzad moeinzad 66 Apr 16 18:33 3
```

The ‘result.sing’ file is the evaluation of haplotypes which are assembled from the reads. The two columns indicate the ‘Match’, ‘Mismatch’ of the overlaps between phased haplotypes and 454 reads. In order to investigate the overlaps further, one can take a look at the files named ‘/*/ranbow.single.eval’. The following lines are selected from

'/0/ranbow.single.eval' file as an example. Each line shows the similarity and dissimilarity of the overlap between a 454 read and an assembled haplotype in one block. The numbers in parentheses show the similarity and dissimilarity of the overlap. The arrow indicates which haplotype is assumed to be sampled from the same chromosome that the 454 read is also sampled. For instance, (8, 1) indicate eight allele matches and one allele mismatch between a 454 read and the second assembled haplotype.

```
454 17 -------101010100-------
illumina
10 0000000000000000000011 (5, 4)
10 1000000101010110001110- (8, 1) <---
12 --0001000001010001101-- (7, 2)
12 --01-110100111010010--- (4, 5)
11 -1101100011101101010011 (6, 3)
11 -0010100110001110010--- (4, 5)
```

A.2.6 Ranbow phylo

To infer the evolutionary event of the organism with the aid of haplotypes, `Ranbow phylo` can be employed to extract and tune the alignments from all phased regions and report the topology of the evolutionary phylogenetic tree. All the regions phased into 'P' haplotypes are used for constructing UPGMA (unweighted pair group method with arithmetic mean) trees with the MEGA-Computing Core[65]. `Ranbow phylo` can be executed in parallel mode as well. Therefore the files need to be indexed first.

```
python2_7_13 ranbow.py phylo -par hap.params -mode index
```

`-mode run` for parallelization:

```
python2_7_13 ranbow.py phylo -par hap.params -mode run -processorIndex i
```

For collecting the generated data:

```
python2_7_13 ranbow.py phylo -par hap.params -mode collect
```

then `collect` mode is deployed in order to report the final statistics for different phylogenetic tree topologies.

```
hap/RANBOW > python ranbow.py phylo -par params.txt -mode collect
7 (((((())))))
7 ((((()())))
29 (((())()))
13 (((()))())
43 ((()())())
26 ((())(()))
```

Moreover `ranbow phylo -mode tree` calculates the mutation rates in the branch trees reports the relative branch lengths.

```
hap/RANBOW > python ranbow.py phylo -par params.txt -mode tree
Tree topology:  (((A-B)I (C-D)J ) M (E-F) N)
Branch Name Mean relative distance SD relative distance
A:     0.674418604651      0.813288765118
B:     0.674418604651      0.813288765118
I:     0.639534883721      0.721961003548
C:     0.732558139535      1.05295176025
D:     0.732558139535      1.05295176025
J:     0.581395348837      0.604539345654
M:     0.886627906977      1.60414025402
E:     0.779069767442      0.891186337505
F:     0.779069767442      0.891186337505
N:     1.0523255814       1.2325032897
Mutation rate for A-B and C-D branches:  0.00701902094469
Mutation rate for E-F branch:  0.00848135858809
hap/RANBOW >
```

Appendix B

Dataset Availability

There was no gold standard real dataset for polyploid haplotyping; hence we construct and introduce the following datasets from sweet potato sequencing data. We release a real semi-ground truth dataset for haplotype reconstruction for polyploid genomes. This dataset can be used for evaluating and comparing methods on real data. We used the trimmed Roche 454 reads, which were not used for *de novo* assembly and haplotyping, to evaluate the assembled haplotypes. Since the Roche 454 reads are limited in size, this data set can not be considered as a full gold standard data and that is why we called it semi-ground truth dataset. Most of the haplotypers use the haplotypes and the reads in coded allele space; hence, we converted Illumina and Roche 454 reads into coded allele space. This helps the developers to avoid data conversion and also it reduces the size of the dataset and makes it easier for downloading. We released two datasets one small for selected scaffolds, which were used in *Chapter 4*, and one big dataset which contains all of the scaffolds in the assembly. Here are the links to the mentioned datasets: **Full dataset**

https://owww.molgen.mpg.de/~ranbow/454_all_scfs.frags.bz2

https://owww.molgen.mpg.de/~ranbow/all_scfs.frags.bz2

Selected dataset

https://owww.molgen.mpg.de/~ranbow/454_selected_scfs.frags.bz2

https://owww.molgen.mpg.de/~ranbow/selected_scfs.frags.bz2

https://owww.molgen.mpg.de/~ranbow/selected_scfs.genotypes.bz2

The instruction for using these datasets can be found in:

https://github.com/euginm/CreateFragmentMatrix

Appendix C

Short CV

For reasons of data protection, the curriculum vitae is not published in the electronic version

For reasons of data protection, the curriculum vitae is not published in the electronic version

Appendix D

Zusammenfassung

Diese Dissertation widmet sich dem Problem der Rekonstruktion von Haplotypen in polyploiden Genomen, und der Verwendung der Haplotypen für das *de novo assembly* dieser Genome. Der gewählte Ansatz stützt sich auf *short read* Sequenzierdaten des höchst heterozygoten hexaploiden Genoms der Süßkartoffel. Zunächst wird die Rolle der Heterozygosität und Ploidie im Kontext der Rekonstruktion von Haplotypen durch *short reads* untersucht. Höhere Heterozygosität macht mehr *reads* für die Rekonstruktion von Haplotypen nutzbar, während die Polyploidie das Zusammenfügen der reads in längere Sequenzen erschwert. Dieses Problem wird hier *Ambiguity of Merging Fragments* genannt und durch den beschriebenen Algorithmus Ranbow adressiert. Die Leistung von Ranbow wird mit Hilfe von realen und simulierten Datensätzen des tetraploiden Genoms des Hirtentäschelkrauts (*Capsella bursa-pastoris*) und des hexaploiden Genoms der Süßkartoffel (*Ipomoea batatas*) evaluiert. Der Vergleich mit anderen Methoden zeigt, dass man mit Ranbow die höchste Genauigkeit und die längsten Haplotypen erreicht. Anschließend wird eine Pipeline für das verbesserte *de novo assembly* des Süßkartoffelgenoms präsentiert, die die zuvor errechneten Haplotypen nutzt. Diese neue Methode, genannt *haplo-scaffolders*, deckt mit Hilfe der Haplotypen einen Satz an möglichen Verbindungen zwischen *scaffolds* auf, die zuvor durch die Unterschiede zwischen echten Haplotypen und der Referenzsequenz versteckt blieben. Diese Verbindungen werden aufgedeckt, indem die *reads* den Haplotypen zugeordnet werden und die Verbindungen auf das Referenzlevel übertragen werden. Der Prozess kann wiederholt werden, in dem der *scaffold* Satz aktualisiert wird, um das Genom *assembly* weiter zu verbessern. Es wird gezeigt, dass diese Strategie den N50-Wert und die maximale Scaffold-Länge des Süßkartoffelgenoms signifikant verbessern.

Appendix E

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Diese Arbeit oder Teile davon sind noch nicht in einem früheren Promotionsverfahren eingereicht worden.

Berlin, 29. August 2018

. . . .

(Mohammadhossein Moeinzadeh)

Glossary

Ambiguity of Merging Problem A characteristic of polyploid genomes which prevent assembly of two fragments with identical shared overlap. 18, 53

Fragment A fsequence of coded variants which are covered by an aligned read. 12, 51

Genotype A set of variants at a polymorphic site. 10

Genotype in coded allele space A genotype is an unordered set so the illustration should include this property; hence, the genotypes are shown as a set of numbers, each corresponding to one allele, separated by slashes. 13

Haplotype The set of variants that are located on a single chromosome. 7

Haplotype-Improved assembly The assembly that is improved via assembled haplotype information.. 103

Haplotype Block A set of haplotype segments with the following properties: 1) Initiated from one mask. 2) The haplotype segments are constructed from the supporting fragments of the seed sequences of the mask. 3) The haplotype segments are elongated by neighboring fragments.. 57, 62

Haplotype segment A haplotype segment is a consensus sequence of a set of fragments. The fragments are supposed to be originated from one haplotype. 53

Haplotypes in nucleotide space A haplotype shown as a sequence of nucleotides. 12

haplotypes in coded allele space A sequence of coded alleles at heterozygous polymorphic sites. 10, 12

Heterozygous variant The sequence variants which are not identical among the homologous chromosomes of a single individual. 10

Homologous chromosomes A group of chromosomes with a similar length and centromere position, which possess similar genes at corresponding loci. 8

Homozygous variant A homozygous variant is a variant which is identical in all homologous chromosomes. 10

Indel Insertion and deletion smaller than 50 base pair. 8

Interpolymorphic region A region between two consecutive polymorphic sites. The sequences of nucleotides in these regions are identical among homologous chromosomes. 8

Mask A mask is an ordered set of indices of polymorphic sites. 54

MEC Minimum Error Correction mode. This model tries to cluster reads and correct their errors such that all reads at each cluster are compatible with each other. 30

Merge function A function that constructs a consensus sequence from a set of input fragments. 53

Minimum Error Correction This model tries to cluster reads and correct their errors such that all reads at each cluster are compatible with each other. 30

Missing allele This terminology, which is illustrated as '−', is used when an allele is unknown in coded allele space. 13

Multinucleotide polymorphism The variants which are larger than one basepair and are equal in length. 8

non-SNP variants Multinucleotide variants and indels. 76

Number of Inferred Haplotypes The number of haplotypes in a phased region. This number could be P like the ploidy of the organism or fewer than P. The latter case could be due to the characteristics of the genome in the region or owing to a failure in sequencing. 89

Polymorphic site A genomic position whose corresponding alleles may vary among the chromosomes in the population. 8

Polymorphism The coexisting of different alleles in the individuals of the same population. 8

Seed sequences of a mask A seed sequence is a subsequence of a fragment excluding missing alleles. It indicates the sequence pattern in one fragment at the indices of the mask. 54

Sequence variant A variant that appears in one chromosome of one individual at a polymorphic site. It could be a single nucleotide variant, a multinucleotide variant, or an Indel. 10

SmP Small Polymorphism, which could be a single nucleotide variant, a multinucleotide variant, or an Indel. 8

SNP Single nucleotide polymorphism, a polymorphic site containing DNA sequence variations with a size of one single base pair whose variant is observed in more than 1% individuals of population. 8

Supporting fragment of a seed sequence A set of fragments which are contributing to a seed sequence. 55

Supporting fragments A set of fragments that is predicted to stem from one haplotype and supports one assembled haplotype segment. 53