Algorithmic designs for reference-based polyploid haplotype phasing

Inaugural-Dissertation

zur Erlangung des Doktorgrades
der Mathematisch-Naturwissenschaftlichen Fakultät
der Heinrich-Heine-Universität Düsseldorf

vorgelegt von
Sven Dominik Schrinner
aus Dortmund

Düsseldorf, Juni 2024

aus dem Institut für Algorithmische Bioinformatik
der Heinrich-Heine-Universität Düsseldorf

Gedruckt mit der Genehmigung der
Mathematisch-Naturwissenschaftlichen Fakultät der
Heinrich-Heine-Universität Düsseldorf

Berichterstatter:

1. Prof. Dr. Gunnar W. Klau

2. Prof. Dr. Tobias Marschall

Tag der mündlichen Prüfung: 10.02.2025

Statement

I declare under oath that I have produced my thesis independently and without any undue assistance by third parties under consideration of the “Principles for the Safeguarding of Good Scientific Practice at Heinrich Heine University Düsseldorf”.

Ich versichere an Eides Statt, dass die Dissertation von mir selbständig und ohne unzulässige fremde Hilfe unter Beachtung der “Grundsätze zur Sicherung guter wissenschaftlicher Praxis an der Heinrich-Heine-Universität Düsseldorf” erstellt worden ist.

Düsseldorf, Juni 2024

Sven Dominik Schrinner

Abstract

The DNA of complex organisms like animals or plants is organized in multiple chromosomes, each carrying a part of the hereditary information. Most organisms carry more than one copy of each chromosome, which we call haplotypes. While humans are diploid and thus carry two copies, many plant species are polyploid with more copies. The process of determining the exact haplotype sequences for each chromosome is called phasing and has a wide range of applications in clinical research, genome analysis, or plant breeding.

In this thesis, we propose two new algorithms for polyploid phasing: The first algorithm aims at inferring the haplotypes of a single individual for which it uses (i) short DNA sequences – called reads – that have been obtained by sequencing machines from a sample of cells and (ii) a reference genome of the target species that has to exist prior to the phasing. We give an in-depth description of our method and compare it to other state-of-the-art methods in the polyploid field. We show that our algorithm is competitive on a variety of different data and is – to our knowledge – the only one that can track uncertainty in the output.

The second algorithm also requires a reference genome but uses genotype information from two parents and a large panel of offspring samples to infer the parental haplotypes. Computing long contiguous haplotypes is challenging due to the limited length of available reads. In contrast, the transmission of parental haplotypes to common offspring samples via Mendelian segregation provides more reliable long-range information on genetic variants co-occurring on the same parental haplotype. We provide a proof-of-concept for our method by showing its accuracy on a few selected regions of the potato genome for which we were able to derive a high-quality phasing from a HiFi assembly. Additionally, we describe a hybrid approach that utilizes both sequencing data and genetic information to combine the strengths of both methods and give an outlook for further research in this area.

Lastly, we present a heuristic for the existing diploid phasing algorithm WHATSHAP that also combines read and genotype information from related individuals but needs to downsample the read data if it is too large. We show that our heuristic is competitive with the exact model in terms of runtime and phasing quality and can slightly outperform it on one out of two datasets when avoiding the strict downsampling.

Kurzfassung

Die DNA von komplexen Organismen wie Tieren und Pflanzen ist über mehrere Chromosome verteilt, von denen jedes einen Teil des Erbguts in sich trägt. Viele Organismen besitzen mehrere Kopien jedes Chromosoms, sogenannte Haplotypen. Während Menschen diploid sind und somit zwei Kopien besitzen, sind viele Pflanzen polyploid und besitzen mehr Kopien. Das Verfahren, um die genauen Haplotyp-Sequenzen zu bestimmen, heißt Phasing und findet breite Anwendung in der klinischen Forschung, der Genomanalyse und der Pflanzenzucht.

In dieser Dissertation stellen wir zwei Algorithmen für polyploides Phasing vor: Das Ziel des ersten Algorithmus ist die Berechnung der Haplotypen eines einzigen Individuums, für die wir zum einen kurze DNA-Fragmente (Reads) verwenden, die von Sequenziermaschinen aus Zellproben gewonnen werden, und zum anderen ein Referenzgenom der betrachteten Spezies, welches zuvor durch andere Methoden bestimmt werden muss. Wir beschreiben unsere entwickelte Methode ausführlich und vergleichen sie mit anderen Referenzalgorithmen für polyploides Phasing. Wir zeigen, dass unser Algorithmus auf einer Reihe von verschiedenen Datensätzen kompetitiv ist und zudem als – nach unserem Wissen – einzige Methode die Angabe von Unsicherheiten im Ergebnis erlaubt.

Der zweite Algorithmus basiert ebenfalls auf einem Referenzgenom, aber benutzt Genotyp-Informationen von zwei Eltern und einer großen Population direkter Nachkommen anstelle von Reads, um die Haplotypen der Eltern zu bestimmen. Während es wegen der begrenzten Länge der Reads schwierig ist, lange und durchgehende Haplotypen zu berechnen, erlauben die Mendel'schen Vererbungsregeln die Identifikation von genetischen Varianten (die auf dem gleichen Eltern-Haplotypen vorkommen) auch über wesentlich größere Entfernungen. Wir zeigen, dass unsere Methode grundsätzlich akkurate Ergebnisse produzieren kann, indem wir sie für einige ausgewählte Regionen des Kartoffelgenoms mit einem Assembly aus HiFi-Reads validieren. Darüber hinaus beschreiben wir, wie beide Algorithmen zu einem hybriden Ansatz kombiniert werden, welcher die jeweiligen Stärken vereint, und erörtern Möglichkeiten für weitere Forschung in diesem Bereich.

Abschließend präsentieren wir eine Heuristik für den existierenden diploiden Algorithmus WHATSHAP, der ebenfalls Reads und Genotyp-Informationen von verwandten Individuen kombiniert, aber nur einen Teil der Read-Daten verarbeiten kann, wenn diese zu groß sind. Wir zeigen, dass die Heuristik kompetitiv bezüglich Laufzeit und Phasing-Qualität ist und den exakten Algorithmus in einem von zwei Datensätzen leicht übertreffen kann, wenn alle Read-Daten genutzt werden.

Acknowledgements

First and foremost, I would like to thank my supervisor Prof. Gunnar Klau, who guided me through my journey with his knowledge and experience. Without his encouragement, especially in stressful times, my scientific work would not have been possible. In that regard, I would like to extend my gratitude to Prof. Tobias Marschall for sparking my research project and providing invaluable ideas and discussions. Next, I would like to thank all my colleagues that I met during my time at the AlBi group. In chronological order, this includes Martin Engler, Eline van Mantgem, Philipp Spohr, Nguyen Khoa Tran, Max Ried, Laura Kühle, Daniel Schmidt, Sarah Schweier, and Sara Schulte. Next to the scientific discussions, there was always some time for nerdy or fun stuff, which created a great atmosphere to work in. I am also very grateful to Prof. Petra Mutzel, who supervised my Master's thesis at my old university in Dortmund, encouraged me to stay in academia, and put me in touch with my future supervisor. Without her advice, I might not even have applied for my PhD project. Lastly, I would like to thank Jens Quedenfeld, John Maleki, Henning Stute, Nguyen Khoa Tran, Philipp Spohr, and especially Laura Kühle for proofreading my thesis and offering helpful suggestions.

Contents

Introduction

The deoxyribonucleic acid or DNA is a large molecular structure that is present in almost all life forms. It carries the hereditary information that is necessary for the function and reproduction of organisms [1]. Each DNA molecule is formed by two parallel strands of nucleotides that are intertwined to form a double helix structure. The nucleotides of each strand encode the genetic information of the DNA through one of four nucleobases – adenine (A), cytosine (C), guanine (G), or thymine (T) – residing on each of the nucleotides. The strands are connected through hydrogen bonds between the nucleobases, where adenine and thymine form one possible binding pair and cytosine and guanine form the other. Since each base in one strand only has one binding partner, the two strands are complementary and both contain the entire genetic information.

The DNA of eukaryotes (i.e. animals, plants and fungi) is arranged in multiple packages that we call chromosomes. Each chromosome encodes a different part of the DNA and might consist of multiple homologous copies that are referred to as haplotypes. The exact number of haplotypes per chromosome – also called ploidy – depends on the species. Humans and most mammals, for instance, are diploid, i.e., every chromosome consists of two haplotypes, while many plant species carry more haplotypes, making them polyploid organisms. Individuals of the same species usually possess the same number of chromosomes and exhibit a high similarity in their DNA sequences which separates them from individuals of other species but still allows for different properties and appearances among individuals of the same species.

Genetic variations and their consequences are the subject of many research fields, as they can be the cause of many diseases, especially hereditary ones and cancer. However, genetic variation is not only observed between different individuals but also between the haplotypes within the same individual. This variation arises from reproduction, where each haplotype inherits different traits from each of the two parents, and from mutations that can randomly occur in the heredity process, from DNA replication or from external events. Determining which genetic variation exists at a particular locus of the DNA among all haplotypes is called genotyping.

A key step to accomplish genotyping is DNA sequencing, the extraction of the nucleobase sequences from DNA molecules through sequencing machines. Over the last decades, a variety of sequencing technologies with different properties has been established. All of them fragment the DNA molecules from a provided sample into smaller pieces to produce short

subsequences of the contained DNA, known as reads. Their length and accuracy greatly depends on the used sequencing technique, making different techniques preferable for different applications.

One of those applications is the reconstruction of the homologous DNA copies, known as haplotype phasing. It poses a more advanced step than genotyping, because it does not only aim to determine the present variation on each locus, but also resolves which variation co-occurs on the same haplotype. This information is essential for a complete understanding of the connection between the genetics and observable traits and characteristics of an individual, also called phenotype. The interaction of genetic variations depends on their distribution among the haplotypes and can lead to vastly different phenotypes even when the genotype is identical [2]. This makes haplotype phasing an important tool for genome analysis and clinical research [2, 3]. In addition, knowledge of haplotypes also enables research in population genetics, like demographic history [4], selection in human population [5] or finding founder sets [6, 7] to which all genetic sequences of a population can be traced back through a series of crossovers.

Next to human genetics, plant genomics also gained more attention over the past few years [8]; the genomes of many important food crops like potato (*Solanum tuberosum*), bread wheat (*Triticum aestivum*) and durum wheat (*Triticum durum*) are polyploid, which makes their reconstruction more challenging, especially since they also prove to contain much more genetic diversity than found in human genomes. Still, resolving polyploid genomes at haplotype level is crucial for understanding the evolutionary history of polyploid species: Evolutionary events, such as whole genome duplications, can be traced back and reveal the ancestry of polyploid organisms [9]. Beyond that, knowledge of haplotypes is key for advanced breeding strategies or genome engineering, especially for improving yield quality in important crop species [9, 10, 11]. Independent from applications, ongoing advances in sequencing technologies regularly create the need of novel or adapted phasing methods, keeping this field of research active.

This thesis introduces two phasing methods for polyploid organisms; the first method follows the paradigm of using reads to infer the haplotype sequences of a single individual. We point out challenges of polyploid phasing, as well as possible shortcomings of existing methods in this field. In contrast, the second method relies on short reads and instead uses a large cohort of offspring samples, bred from the same pair of cultivars; utilizing Mendelian segregation enables the method to identify co-occurrences of genetic variants on the same haplotype over much larger distances than would be possible by sequencing data from one individual alone. The drawback is that the approach is limited to certain types of genetic variants. These properties complement those of the first method and hence we discuss how these ideas can be combined to harness the strengths of both approaches.

Outline

Our work is divided into four chapters: Chapter 1 introduces the relevant biological background, basic concepts, and terminology that is used throughout the rest of the thesis.

In Chapter 2, we review an existing diploid phasing method for single individuals, called WHATSHAP [12], that is based on the commonly used Minimum Error Correction (MEC) model. Since this model is not applicable for data with high read coverage, and thus relies on downsampling input data, we propose a heuristic version that is able to utilize larger datasets at the cost of losing optimality with respect to the underlying MEC model. Analogously to the work of Garg et al. [13], we also present an extension to phase multiple related individuals. Both heuristics have not been published before.

The first of two novel polyploid phasing methods is introduced in Chapter 3. We originally published this method in 2020, where we discussed caveats of polyploid phasing and compared our method to existing state-of-the-art ones [14]. Since the development of this method continued afterwards, we will present the most recent version that features some key differences to the original one. It is based on read clustering and a novel approach that we call *haplotype threading*. We reproduced our original evaluation and added both new datasets and an additional phasing tool that has been released in the meantime [15]. Our original work, on which I share first authorship with Jana Ebler and Rebecca Serra Mari, was published in *Genome Biology*. My main contribution was the development and implementation of the read clustering. I was the main contributor to all consequent development that has not been published separately.

In Chapter 4, we present a second polyploid phasing method with a different scope of applications: Instead of processing single independent samples, its purpose is to phase two parental samples (of the same species), for which we have access to a large panel of progeny sequencing data. Combining data from related individuals facilitates the use of long-range information, induced by Mendelian heredity rules. In the context of plant breeding, it is relatively easy to create many offspring samples from the same two parents. Sequencing such a large offspring pool offers more statistical power than working with trios of two parents and a child, like it is done for humans. Thus, our genetic polyploid phaser is centered around a statistical model to find co-occurrences of certain marker alleles on the parents' haplotypes by using low-depth sequencing data from a large offspring population. We show in a proof-of-concept that our method is able to compute sparse phasing based on short-read data and a reference genome only. The work has been published in *iScience* [16].

Chapter 1

Preliminaries

This chapter introduces the biological background that is needed to understand the concepts presented in this thesis. It establishes a theoretical framework shared by all haplotype phasing methods described in Chapters 2, 3, and 4. If not stated otherwise, each section has been newly compiled for this thesis.

The goal of this thesis is to present novel techniques for haplotype phasing and discuss the challenges of polyploid genomes for this field. We already introduced DNA as the carrier of hereditary information in organisms and the importance of studying its structure and function. In this chapter, we will define and explain basic concepts and terminology on which our presented methods are based. First, we will discuss properties of DNA and how we can extract the genetic information from a given sample such that we can process it using computational methods. Second, we will review different types of haplotype phasing to embed our methods in this field of research. And finally, we will formulate haplotype phasing as a mathematical problem that can be solved computationally.

1.1 DNA sequencing and mapping

In order to analyze the DNA of specific samples, we need to extract the genetic information from its cells. On an abstract level, this information is encoded as long sequences over the four nucleobases (A, C, G, and T), with one sequence for every copy of each chromosome. The process of extracting sequences of nucleobases directly from DNA molecules is called *sequencing*. Many downstream analyses – including haplotype phasing – require a digital representation of DNA and DNA fragments for processing and visualization. Therefore, we consider the DNA (or parts of it) as a long string over the nucleobase alphabet. Since the two strands of a DNA molecule contain complementary base pairs, it is sufficient to encode one of the strands as a string.

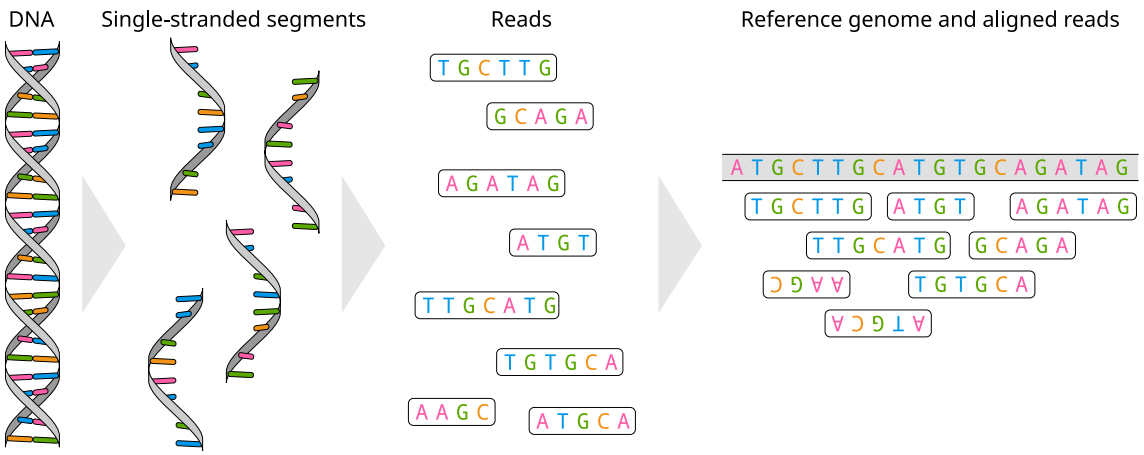


Figure 1.1: **Read sequencing and alignment.** The process starts with a double-stranded DNA molecule that contains two complementary sequences of nucleobases (indicated by colors). This molecule is fragmented into single-stranded segments, from which reads are randomly sampled. All reads are then aligned against a (previously obtained) reference genome. The bottom two reads have been sequenced from the complementary strand, and are thus aligned as *reverse complements* with base order reversed and each base replaced with its complement.

1.1.1 Sequencing technologies

DNA sequencing is a technique with numerous applications that has been evolving over many decades by now. Sanger sequencing [17] and the Maxam-Gilbert chemical cleavage method [18] are considered the first generation of sequencing methods and were published around 1976. In the late 1990s to early 2000s, followed so called next-generation sequencing technologies, while third-generation sequencing describes the technologies introduced from the mid-2010s until today. Still, until today none of the available sequencing technologies is able to extract genetic information on a chromosome level. Instead, all technologies produce small DNA fragments, called *reads*, each of which contains a short subsequence of one of the chromosomal copies inside a sampled cell. Additionally, reading the actual bases from the DNA fragments is error-prone. Thus, each read has a chance to contain *sequencing errors*, which can be wrongly called bases, missing bases, or wrongly inserted bases, e.g., reading a single base twice. In this thesis, we only give a very short overview of sequencing technologies that are relevant to our presented methods. For a more detailed survey of the history of sequencing [19, 20, 21] or the functionality of different technologies [22, 23], we refer the reader to more specialized literature in this field.

Short-read sequencing

Today, we generally differentiate between two kinds of sequencing data: short- and long-read sequencing. The former is often used as a synonym for next-generation sequencing. Its key difference to previously existing technologies is the ability to parallelize the steps for sequence synthesis on a large scale to achieve significantly higher throughput. A prominent example of this technology is Illumina sequencing, which produces reads of length 75–300bp (base pairs, i.e., the number of consecutive bases), depending on the exact type of used sequencing machine [24]. The accuracy of these reads is very high with a per-base error rate of $<1\%$ in

practice [25, 26]. Illumina sequencing also offers paired-end reads, i.e., a single piece of a DNA strand is sequenced from both ends, resulting in a pair of short reads that is guaranteed to stem from the same chromosomal copy with a rough estimate of the gap length between them (usually less than 1000bp).

Before the sequencing procedure, the DNA molecules are randomly fragmented into small pieces of single strands, from which the final reads are extracted. The steps from a full DNA molecule to sampled reads are illustrated in Figure 1.1. Since the DNA fragmentation is random, each read can either be sampled from the forward strand or the backward strand, which has to be accounted for in all following steps, like *read mapping*.

Long-read sequencing

More recent sequencing technologies like Pacific Biosciences CLR (PacBio) and Oxford Nanopore Technologies (ONT) use different biochemical approaches and produce much longer reads than next-generation sequencing. This distinction led to the term *long-read sequencing* or *third-generation sequencing*. PacBio reads typically have a length between 10–60kbp (1kb = 1000bp) with an average of about 15kbp and an error rate of 8–15% [23, 27]. ONT reads are slightly longer with 10–100kbp and an error rate between 2–13% [23]. In 2019, PacBio presented an improved technique based on their single-molecule technology using *Circular Consensus Sequencing (CCS)* [28]. It sequences the same isolated DNA fragment multiple times in a circular manner to eliminate errors by forming a consensus over all runs. The reads are comparable in length to the original PacBio reads with about 10–30kbp, but feature short-read-like error rates of <1% (0.2% on average).

1.1.2 Read mapping

To ensure that all genome positions are covered and to compensate for possible sequencing errors, a sequencing run typically produces many more total bases than the target genome has. This means that every genome position is covered by more than one read and that some of the reads overlap in terms of genome position. The number of reads covering each position on average is called the *coverage* of the readset. Typical coverages when sequencing an organism are around 30× to 60×, depending on sequencing technology and species, but can also greatly deviate from this range if required by the application.

The original position of each read within the genome is unknown after sequencing. One possibility to determine their position is to compute genome assemblies from the reads: Read pairs with a sufficiently large overlap can be merged into longer continuous sequences, called *contigs*, to eventually span larger parts of the chromosome. However, read-based genome assembly is computationally expensive and often fails to fully reconstruct entire chromosomes due to repetitive regions in the genome.

An alternative to finding read positions inside the genome is to use a *reference genome* for the target species. A reference genome is a set of DNA sequences – one for each chromosome

– that acts as a blueprint for the genome of a specific species. It usually does not represent a single individual but rather an ensemble of multiple ones. Since the number of chromosomes is identical between all individuals of the same species (except for genomic disorders) and the placement and shape of all genes only differ slightly, a reference genome is a useful pivot for locating most of the reads on the genome, even though a single reference sequence is unable to capture genetic diversity.

Mapping reads to their most likely position in the genome is done through sequence alignment. Since reads contain both sequencing errors as well as genetic variation, they cannot be mapped with an exact string search over the reference genome. Most approaches are based on the Needleman-Wunsch [29] or Smith-Waterman algorithm [30] for global and local sequence alignment, respectively. Some tools use seeding strategies based on k-mers (substrings of length k) or exact matches of small parts of the reads to first find approximate candidate positions for each read before applying a costly alignment. Examples of commonly used tools are BWA-MEM [31] for short reads and Minimap2 [32] for long reads.

1.2 Genetic variation

After the sequencing data has been aligned against a reference genome, we want to identify genetic variation among the sequenced individual. We provide a classification of possible genetic variants in this section and describe how they can be detected based on the aligned reads. Diploid and polyploid organisms possess multiple haplotypes per chromosome, so we additionally want to quantify how often each detected variant occurs among the haplotypes. This process is called *genotyping*.

1.2.1 Types of variation

The most common type of variants are *single nucleotide polymorphisms* (SNPs) [33]. They describe the alteration of a single base in the genome of an individual (compared to a given reference genome). In human genomes, there is more than one SNP per 1000bp on average [33]. Plant genomes can be much more diverse with some potato cultivars reaching more than one SNP per 25bp on average [34].

In *deletion* and *insertion variants*, short sequences of bases are either missing in an individual genome or newly inserted, i.e., the individual contains an additional sequence that is not present in the reference. These two variant types are often summarized as *indels*. The classification as indel is restricted to deleted or inserted sequences of limited size, usually 50bp. If a longer sequence is deleted or inserted, such variants are rather classified as *structural variants* or long indels. This distinction follows from the fact that large variants are much harder to detect than short ones because reads become harder to map to a reference genome if they contain many or large variants.

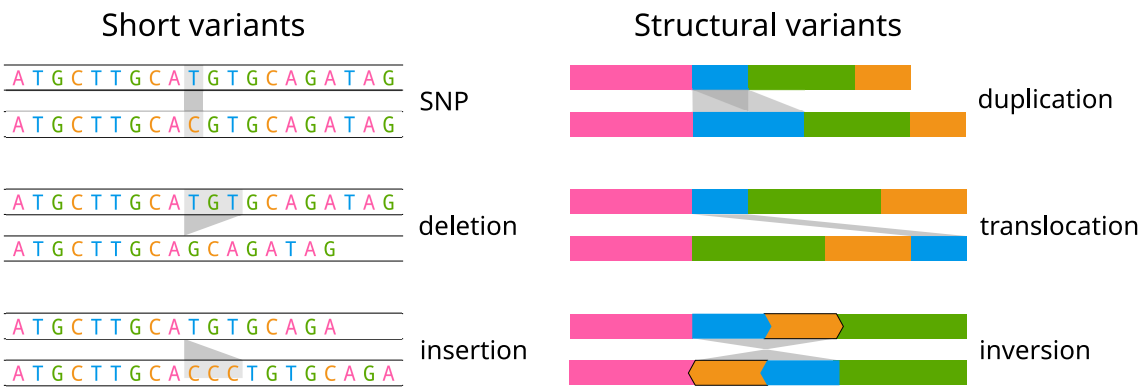


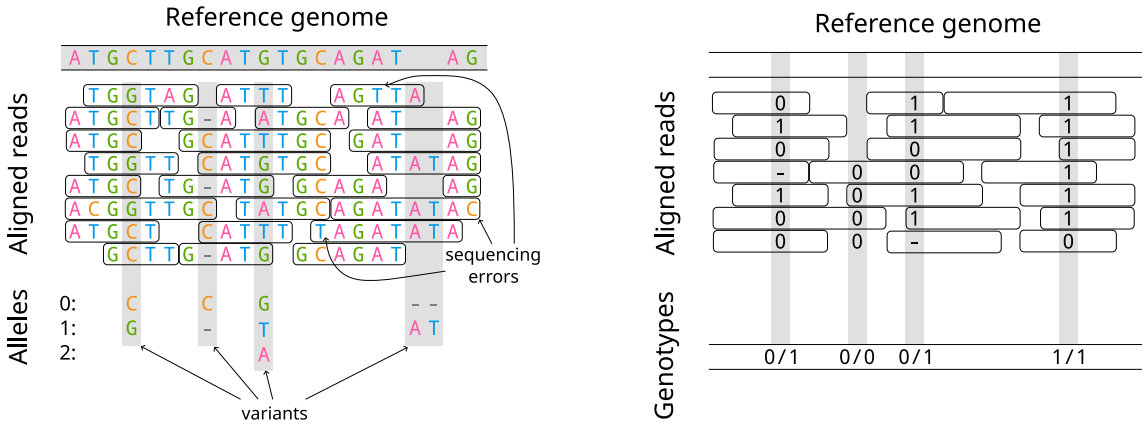
Figure 1.2: **Variant types.** The figure shows six types of genetic variants: SNPs, deletions, insertions, duplications, translocations, and inversions. In each example, the top sequence is the reference and the bottom one the sequence containing the variant. **Left:** A single base is mutated or a short sequence of bases is either deleted or inserted with respect to the upper sequence that acts as a reference here. The gray part highlights the changes between each pair of sequences. **Right:** For structural variants, different homologous genome parts are depicted as segments of the same color. Segments can be translocated or duplicated or a sequence of segments can be inverted. For inversions, the segments have a tipped end to indicate their orientation.

The three small variant types are depicted in Figure 1.2. In addition, it shows three types of structural variants that describe large-scale differences between two genomes: *Duplications* occur when a part of a chromosome is copied and inserted again in the genome. If the copies are adjacent to each other, this variation is also called *tandem duplication*, otherwise, it is an *interspersed duplication*. In case of a *translocation*, a segment of the genome is removed from one locus and re-inserted at a different one. This can even take place between different chromosomes. An inversion event flips a genome segment in place, such that the affected sequence is reversed and re-connected to its adjacent segments with both endpoints switched. All these types of structural variants can be combined in numerous ways, e.g., as an inverted duplication, where the duplicated sequence is inserted in reverse order.

Reference-based methods usually omit structural variants because they are difficult to be assigned to a distinct position on a linear reference genome and overlap with many other variants. A more suitable way to represent structural variation is *genome graphs* that carry genome sequences on each of their nodes and insert directed edges between two sequences when they are adjacent in any of the underlying input genomes or reads on which the graph was built [35]. For this thesis, however, genome graphs are out of scope and we will only focus on reference-based methods.

1.2.2 Genotypes

Genotyping requires an already computed set of variants for the used reference genome. In this context, a *variant* consists of a genome position and a set of possible *alleles*, i.e., a set of possible sequences an individual may possess at the corresponding genome position. In Figure 1.3a, we see an example consisting of a reference genome and four distinct variant positions; the first variant is a SNP with possible alleles `C` and `G`, while the third variant is a SNP with three alleles (`G`, `T`, and `A`). The second variant shows a deletion with alleles `C` and `-`, indicating that individuals might either have a `C` at the respective genome position or no



(a) Variant calling

(b) Genotyping

Figure 1.3: **Variant calling and genotyping.** **(a)** A set of reads is aligned against a reference genome. The vertical gray stripes indicate variant positions, on which significant variation between the reads and the reference is present. The bottom ends of these stripes show the present alleles. Dashes stand for the absence of a base. Rare variations among the reads are considered to be sequencing errors. **(b)** This figure shows the same four variants as in (a), but with a different set of aligned reads. For each read and variant, the index of the most likely allele is noted down; if the read does not support any of the alleles we denote this as "-". The bottom part states the most likely genotypes for the diploid individual.

base at all. Analogously, the fourth variant is an insertion that considers individuals to have an optional AT-sequence at the respective genome position.

The set of variants is either computed from the reads of the individual to be genotyped or might be based on previous computations from possibly multiple individuals. For the so-called *variant calling*, we utilize aligned reads to detect statistically significant deviations from the reference genome. The challenge is to differentiate between sequencing errors on the reads and true genetic variation. The former are distributed randomly (with certain technology-specific biases), while the latter result in a systematic deviation between a subset of reads and the reference. Therefore, a sufficiently high coverage is needed to reliably identify variant positions and existing alleles. There exist several tools for variant calling, like FreeBayes [36], GATK [37], or Varlociraptor [38], each of which is based on different computational models.

In genotyping, the variant positions and alleles are known and the goal is to find the most likely combination of alleles among the haplotypes of an individual for each variant. These combinations are called *genotypes*. The number of alleles per variant is given by the ploidy of the genotyped organism. Genotypes are usually represented as integer lists, separated by a "/" and sorted in ascending order. The integers encode indices of existing alleles for each variant. By convention, the allele following the reference genome (also called reference allele) is given the index 0, while the alternative alleles are continuously indexed as $1, 2, 3, \ldots$. Figure 1.3b shows an example of a diploid individual with four genotypes for the four existing variants. If all alleles of a genotype are identical, it is called *homozygous* (like the second and fourth variants); otherwise it is called *heterozygous*.

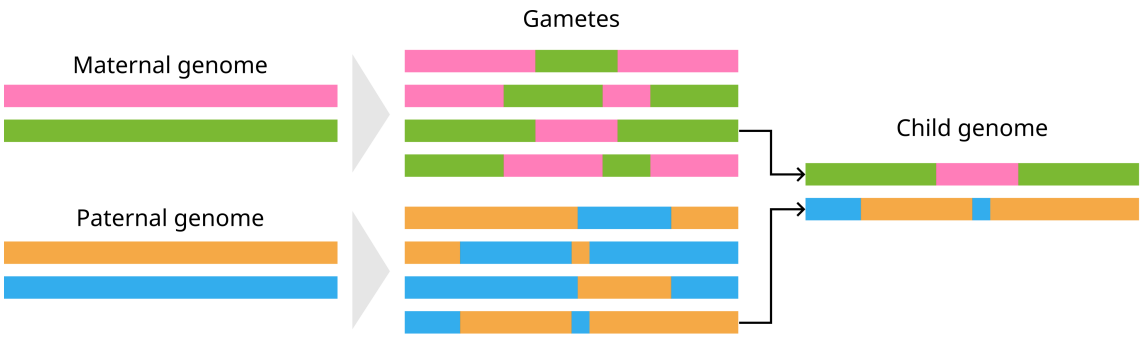


Figure 1.4: **Recombination.** Each of the two diploid parents produces a total of four gametes during meiosis. The color distributions inside the gametes indicate the recombination events between different haplotypes. The diploid child genome is formed by one gamete from each parent, which are selected at random.

1.2.3 Reproduction

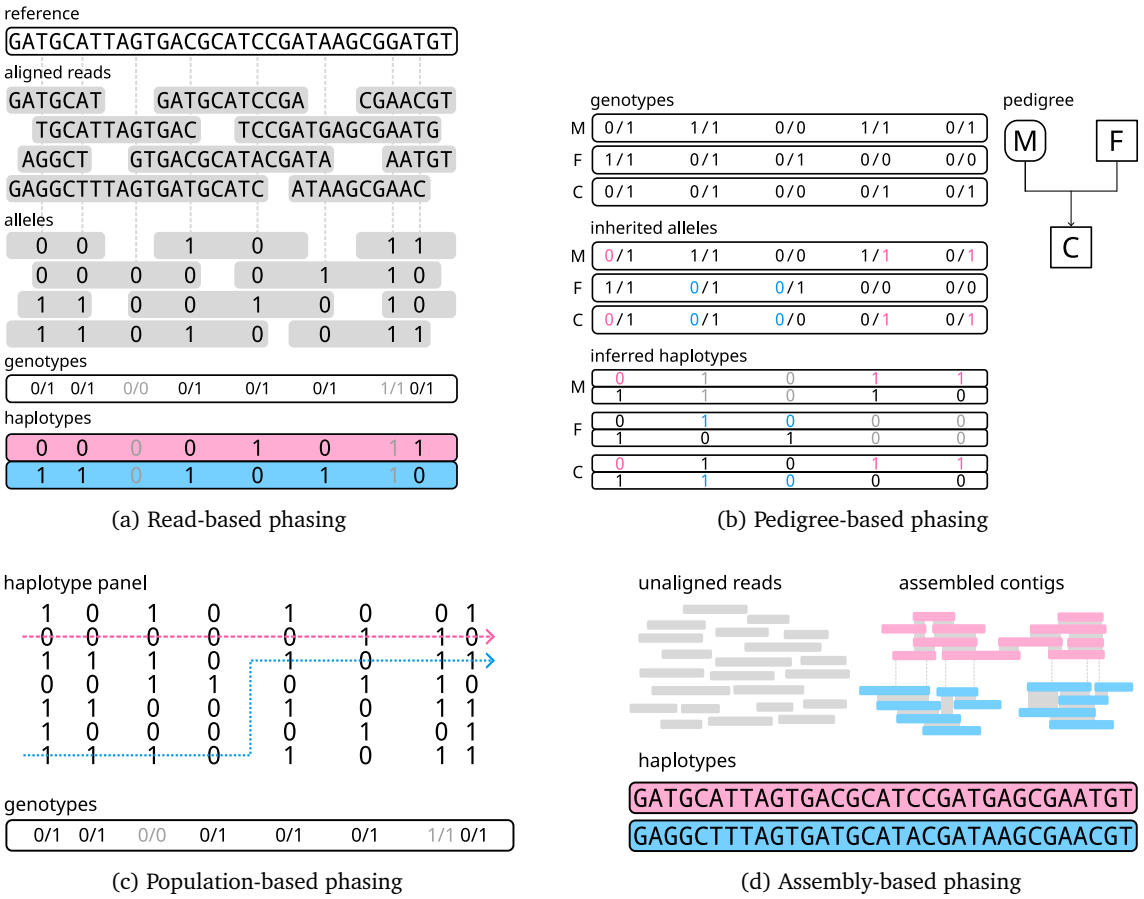
The process of sexual reproduction involves a special type of cell division, called *meiosis* [1]. While regular cell division results in two identical cells with the same number of haplotypes, meiosis produces four gametes with only half as many haplotypes. It replicates the genetic material inside a cell but additionally includes a recombination step in which haplotype fragments are exchanged between the two chromosomal copies. After division into two cells with a full set of (recombined) haplotypes each, these sets are again divided into four gametes that contain unique mosaics of the individual's original haplotypes. The fusion of one maternal and one paternal gamete results in a complete genome with each half of its haplotypes containing genetic material from the mother and father, respectively. In Figure 1.4, the process is illustrated for a diploid organism. The recombination events inside the gametes are indicated by color shifts.

1.3 Haplotype phasing

Haplotype phasing describes the process of reconstructing the individual sequences for each chromosomal copy. That means that the genome of a target individual is resolved to a haplotype level. There exist multiple approaches for haplotype phasing, each using different kinds of data and methods. These approaches mainly fall into three categories: Read-based phasing (also called molecular phasing), pedigree-based phasing, and population-based phasing [3, 39]. All of these methods rely on a reference genome as a pivot for their computation. If a reference genome is not available or one wants to avoid any bias induced by such, there are also so-called genome assembly methods that reconstruct genomes without a given reference. Thus, one could refer to reference-free approaches as a fourth category of haplotype phasing, if the assemblies are resolved on a haplotype level.

1.3.1 Read-based phasing

A widely used reference-based approach is read-based phasing, which requires sequencing data from the phased individual, as well as a (linear) reference genome for the target species.



(a) Read-based phasing

(b) Pedigree-based phasing

(c) Population-based phasing

(d) Assembly-based phasing

Figure 1.5: **Categories of haplotyping methods.** **(a)** Read-based phasing with reads aligned to the reference genome. All reads are reduced to their allele indices for each variant position (vertical dashed lines): Haplotype sequences are inferred from read-induced connections between alleles. **(b)** Pedigree-based phasing with a trio of mother, father, and child: From genotypes and Mendel's law, inherited alleles can be tracked down (magenta and blue fonts), as a parent cannot pass on an allele to the child if it does not possess it. Assuming no recombination, parent haplotypes can be phased alongside the child haplotypes. **(c)** Population-based phasing, based on a haplotype panel and genotypes of a new sample: For haplotype inference, the set of paths best explaining the input genotypes is computed. **(d)** Reference-free assembly of haplotypes by building contigs of overlapping reads.

In a preceding step, all sequenced reads are aligned against the reference genome to identify heterozygous sites and their genotypes. Each read that covers at least two heterozygous sites is considered informative for the phasing process, while all other reads are discarded. Considering that every read originates from a single DNA molecule, the kept reads carry information on which alleles co-occur on the same haplotype. Figure 1.5a depicts these steps with the addition of representing reads and haplotypes as sequences of integers over variant positions only. This is a common way in haplotype phasing to reduce input and output data to its relevant core. We will introduce this notation more formally in Section 1.4.

While short and accurate reads are often preferred for variant calling or genotyping, read-based phasing greatly benefits from long reads. The reason is that the main challenge is to provide a contiguous, accurate prediction of the underlying haplotypes that does not contain wrongly joined haplotype fragments (misassemblies) or is broken into many small pieces because of missing connections between certain variant sites. These errors are usually caused by long gaps between heterozygous variants or difficult genomic regions (e.g. repeats) aggravat-

ing read alignment. Therefore, read length is considered the main bottleneck, whereas high error rates rather lead to isolated mistyped alleles with low impact on overall correctness.

There exists a variety of phasing algorithms, each of them using the read information in different ways and solving different optimization models. Some notable mentions are WHATSHAP [12] for diploid phasing and H-POP-G [40], WHATSHAP POLYPHASE [14], and FLOPP [15] for polyploid phasing. A more comprehensive list of available methods and their concepts will be given in Chapters 2 and 3.

1.3.2 Pedigree-based phasing

Pedigree-based phasing utilizes the hereditary relationship between haplotypes of parents and children. This approach can be combined with sequencing data, although this is not mandatory. The prerequisites are again a linear reference genome for which the heterozygous sites of an entire family of individuals and their respective genotypes are known. Even without reads, it is possible to reconstruct large parts of the individual's haplotypes when provided with a sufficiently large family tree [41]. Since recombinations are considered rare events, a proper phasing model should be parsimonious, i.e. explain the observed genotypes with the least amount of recombination events (and the least amount of germline mutations). A small example of read-less phasing of three related samples is shown in Figure 1.5b.

When using reads, pedigree-based phasing works like an extension to read-based phasing, where all related individuals are phased simultaneously. The difference to independent phasing is the hereditary constraints between the computed haplotypes. The advantage of pedigree-based phasing over (pure) read-based phasing is the long-range information provided by the biological process of heredity. Even when using just trios (mother, father, and a single child), complementing sequencing data with hereditary information leads to greatly superior phasing performance compared to read-based phasing alone [13]. The read-based phasing tool WHATSHAP [12] is also able to work on pedigree data. Pedigree-phasing on polyploid organisms has been done by TriPoly [42], PopPoy [43], and WHATSHAP POLYPHASE-GENETIC [16], which we will introduce in Chapter 4.

1.3.3 Population-based phasing

In population-based phasing, also called statistical phasing, a large reference panel of haplotypes is used as a template for inferring a sample's haplotypes. The rationale is that long haplotype pieces are shared between a cohort of common ancestry, even if the individuals behind it are not directly related to each other [44]. Therefore, a new individual's haplotypes might be formed by recombining previously computed haplotypes based on additional data (e.g. sequencing or genotype data of the new individual). Despite working well for common genomic variants, the drawbacks of this method are that such reference panels only exist for a few species and that rare variants not present in the panel cannot be correctly predicted. Examples of population-based phasing tools are Beagle [45], Eagle [46], and SHAPEIT [47].

1.3.4 Haplotype assembly

The three formerly presented categories rely on a linear reference genome because they need a well-defined set of heterozygous variant sites that is shared between different samples. This linearity reduces haplotype phasing to the problem of selecting the correct alleles for each of the defined variant sites. Aside from requiring a high-quality reference genome in the first place, the major limitation here is that the methods following this paradigm are only applicable to short-range variants, like SNPs and short indels. Haplotypes with too much structural divergence from the reference genome cannot be represented by the scheme of succinctly defined variant positions in a sensible way. Structural variants or rearrangements in the human genome are an important field of study and even identifying large-scale variants is a challenging task [48, 49]. In polyploid organisms, structural variants are even more prevalent as shown by various assemblies for potato cultivars [34, 50, 51, 52].

To overcome the dependency on reference genomes, one might consider assembly-based approaches as an additional category for haplotype phasing [53]. Genome assembly aims to construct genomic sequences of an individual directly from sequencing data, without the help and the bias of an existing reference. This is usually accomplished by identifying overlaps between reads or k-mer-based methods, like Canu [54], Flye [55], or Hifiasm [56]. While classic genome assembly seeks to obtain a single sequence representing the sample's genome, it can also be done on a haplotype level, where each of the chromosomal copies is reproduced individually. Examples of such haplotype assembly approaches are trio binning [57] and FALCON-Phase [58] for diploid assemblies, and a polyploid method by Serra Mari et al. [52]. Assembling haplotypes without a reference genome as a pivot is considerably harder and usually requires additional data, like Hi-C data [58] or sequencing data from related samples [52, 57].

1.4 Terminology and notation

At the beginning of this chapter, we noted that many applications for read sequencing are computational methods that require a formal representation of DNA, reads, and genomic variations. Thus, we will use various terms and notations to present our phasing methods. While some of them are specific to certain steps, a large portion is shared between all of them and we will introduce them throughout this and the following sections. We also refer to the list of variables and symbols in the appendix, where we collect all commonly used variables.

Variants

For all phasing problems, we assume that we are given a specific chromosome that is to be phased. We also assume that a reference genome for this chromosome is available that acts as a coordinate system: It maps every observed genomic variant to a distinct position. A variant v is defined as a tuple (p, A), where p is its start position on the reference genome and A is a list

of possible alleles, i.e., eligible sequences that may occur among any read, starting at position p. As already explained in Section 1.2.2, the first allele in the list is associated with the index 0 and represents the reference allele. All following alleles are referred to as having indices $1, 2, 3, \ldots$ and are called alternative alleles. Let $\mathcal{V} = \{v_1, \ldots, v_m\}$ be the set of m variants on the given chromosome with $v_i = (p_i, A_i)$. For convenience, we assume that all variants are indexed and sorted by position, i.e., $p_1 < p_2 < \ldots < p_m$. Let $a_{\max} := \max_{1 \leq i \leq m} |A_i| - 1$ be the highest number of alternative alleles for all variants in $varset$. If $\mathcal{V}$ only contains two alleles for each variant, then $\mathcal{V}$ is called *bi-allelic* with $a_{\max} = 1$, otherwise it is *multi-allelic* with $a_{\max} > 1$.

Reads

The original reads that are aligned against a reference genome are sequences of bases. For haplotype phasing, we are only interested in their alleles at variant positions. Therefore we redefine reads to be sequences of alleles, given by their local index: A read r is a sequence of length m over the alphabet $\{-, 0, 1, \ldots, a_{\max}\}$ with $r[i]$ being the i-th character of r. Each character of r represents the supported allele for each of the m variants, i.e., $r[i] = a$, if r contains the allele with index a at the genomic position of the i-th variant. If r does not cover the i-th variant or contains an invalid allele, we set $r[i] = -$.

For simplicity, we assume that all reads are well-defined, i.e., all their characters are either gaps or refer to valid allele indices. The first non-gap position of r is denoted as $s(r)$ and the last non-gap position as $e(r)$. A read r *spans* a variant v_i if $s(r) \leq i \leq e(r)$ and it *covers* v_i if $r[i] \neq -$. A set of n reads forms an allele matrix $\mathcal{A} \in \{-, 0, \ldots, a_{\max}\}^{n \times m}$, where each row contains the allele information of a single read and each column contains the allele indices for a specific variant. The set of all input reads without the matrix structure is referred to as $\mathcal{R}$.

Haplotypes and genotypes

Similarly to reads, we redefine haplotypes to be sequences of length m over the alphabet $\{-, 0, 1, \ldots, a_{\max}\}$. Thus, each haplotype represents a combination of alleles over all m variants. The *ploidy* is the number of chromosomal copies for the species to phase and is denoted as p. The set $\mathcal{H} = \{H_1, \ldots, H_p\}$ represents the *true haplotypes* that are unknown to the phasing algorithm but used in evaluation metrics. The predicted haplotypes are denoted as $\tilde{\mathcal{H}} = \{\tilde{H}_1, \ldots, \tilde{H}_p\}$. Haplotype sets possess no order and are only indexed for convenience.

For a variant $v_i = (p_i, A_i)$, the genotype $G(v_i)$ of v_i (abbreviated as G_i) is a multiset of size p over the values $0, \ldots, |A| - 1$. It represents the multiset of present alleles for each variant, usually in concordance with the sequences given by the haplotypes, i.e., $G_i := \{H_1[i], \ldots, H_p[i]\}$. Let $f_a(G_i) = |\{b \in G_i \mid b = a\}|$ be the number of times allele a occurs in G_i. If $\mathcal{V}$ is bi-allelic, every genotype G_i can be characterized as an integer number between 0 and p, which represents the absolute frequency of the 1-allele in G_i (or simply $f_1(G_i)$).

Some phasing models deal with multiple input samples simultaneously. Here we have true haplotypes, predicated haplotypes, and genotypes for every sample s. We use the same notation as for single-sample phasing, but add the sample index to the superscript, i.e., H^s, $\tilde{H}^s$ and G_i^s, respectively.

1.4.1 Problem formulation

As stated before, the goal of haplotype phasing is to predict the true haplotypes for an individual with ploidy p. Since we are only given information that is based on sequencing data (including errors), we cannot directly check how close a computed solution is to the true one. Instead, we can only compute a solution that explains the given input data in the best possible way with respect to some previously defined phasing model. For reference-based phasing, a generic problem formulation is given by Problem 1.4.1.

Problem 1 (**Simple version of reference-based phasing**). *Given an allele matrix $\mathcal{A}$ over m variants and a ploidy p, find a set of p haplotypes $\tilde{\mathcal{H}} = \{\tilde{H}_1, \ldots, \tilde{H}_p\}$ of length m, such that $\tilde{\mathcal{H}}$ maximizes a model-specific objective function.*

Defining a suitable phasing model is part of the algorithmic design and we will introduce multiple models for different applications throughout this thesis. We note here that no model can guarantee finding the true haplotypes even when solved to optimality because the input data is based on random sampling. Moreover, parts of the true haplotypes might not be reconstructible from the input due to ambiguities, e.g. if two consecutive variant positions are so far apart that no read covers both of them.

To account for ambiguities, we add a set $\mathcal{P}$ of *cut positions* as an additional output to our generic phasing problem. A cut position $c \in \mathcal{P}$ indicates that the predicted haplotypes are cut between variants v_{c-1} and v_c because there is a lack of confidence in joining the left and right parts of the predicted haplotypes at this locus. The intervals between two cut positions are called *phasing blocks* and represent coherent sections of the predicted phasing. By convention, position 1 is always contained in $\mathcal{P}$.

Since many variant callers also invoke genotyping, we define a sequence of genotypes $G_i, \ldots, G_m$ for the corresponding variants as optional input. The phasing model may constrain its predicted haplotypes to conform to the provided genotypes are only use them as hints.

A second optional input is a *pre-phasing*. Like an actual phasing, a pre-phasing $\overline{\mathcal{H}}$ consists of p haplotypes $\overline{H}_1, \ldots, \overline{H}_p$ that contain gaps. Its purpose is to encode already existing phasing information that might be incomplete but can still be used as guidance when inferring complete haplotypes from reads. Most of the presented models in this thesis do not use this type of optional input, but we present it as part of the generic problem formulation because such information might be available for some applications. Problem 1.4.1 shows the extended problem formulation with optional inputs and both types of output.

Problem 2 (Extended version of reference-based phasing). *Given an allele matrix $\mathcal{A}$ over m variants, a ploidy p, and optionally m predicted genotypes $G_i, \ldots, G_m$ or a pre-phasing $\overline{\mathcal{H}}$. Compute a set of p haplotypes $\tilde{\mathcal{H}} = \{\tilde{H}_1, \ldots, \tilde{H}_p\}$ of length m and a set of cut positions $\mathcal{P}$, such that $\tilde{\mathcal{H}}$ maximizes a model-specific objective function.*

1.5 Evaluation metrics

The description of evaluation metrics is mostly taken from [14]. It was rephrased and some terms were clarified or supplemented by additional formulas. The switch flip rate and the wrong genotype rate were newly added. Some adjustments were made to cover diploid cases as well.

For all of our developed methods, we will provide performance benchmarks throughout this thesis to compare them to other methods or to show a proof of concept. This section serves as an overview of all performance metrics used in our benchmarks. We use the examples from Figure 1.6 to illustrate the different metrics.

Hamming rate

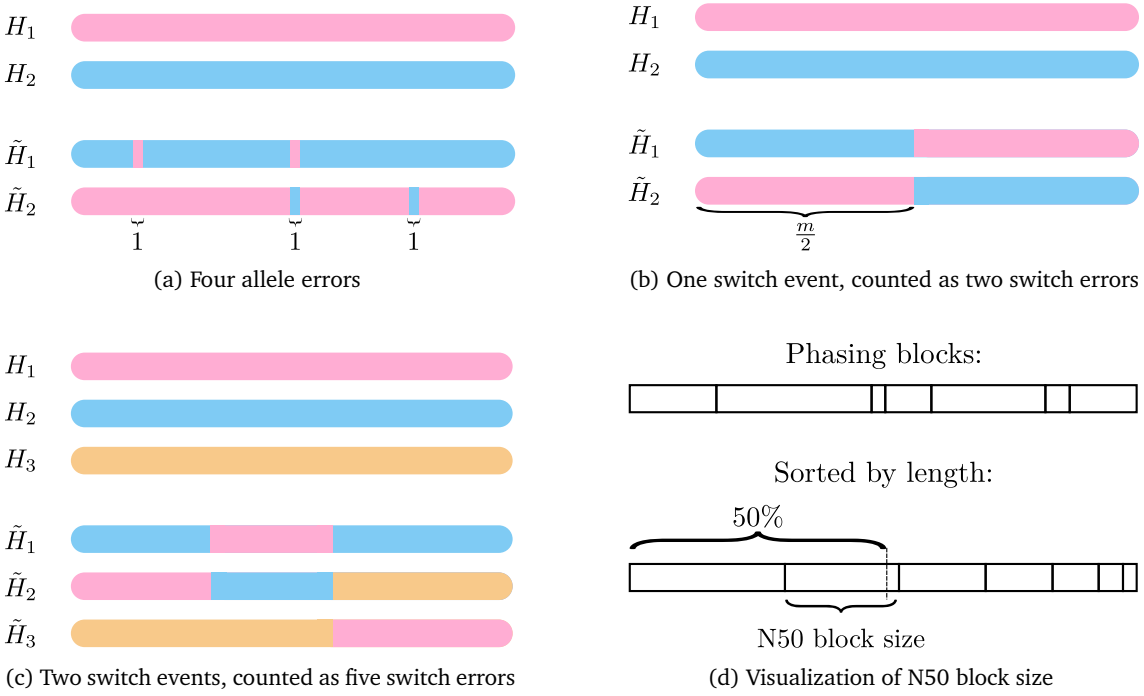
For ploidy p, a set of ground truth haplotypes $\mathcal{H} = \{H_1, ..., H_p\}$ and predicted haplotypes $\tilde{\mathcal{H}} = \{\tilde{H}_1, ..., \tilde{H}_p\}$, we compute the number of *Hamming errors* (HE) as

$$\mathrm{HE}(\mathcal{H}, \tilde{\mathcal{H}}) = \min_{\pi \in S_p} \frac{1}{p} \sum_{i=1}^{p} d_H(H_i, \tilde{H}_{\pi(i)}) = \min_{\pi \in S_p} \frac{1}{p} \sum_{i=1}^{p} \sum_{j=1}^{m} [\![H_i[j] \neq \tilde{H}_{\pi(i)}[j]]\!] \tag{1.1}$$

where S_p represents the permutation group on $\{1, ..., p\}$ and d_H the Hamming distance between two sequences that counts the number of position-wise differences. The Iverson bracket $[\![x]\!]$ evaluates to 1 if the boolean expression x is true and to 0 otherwise. The *Hamming rate (HR)* is then defined as the sum of Hamming errors divided by the total number of all phased variants. If subtracted from 1, the Hamming rate is equivalent to the *reconstruction rate* and the *correct phasing rate* presented in [40] and [59], respectively.

$$\mathrm{HR}(\mathcal{H}, \tilde{\mathcal{H}}) = \frac{\mathrm{HE}(\mathcal{H}, \tilde{\mathcal{H}})}{p \cdot m} \tag{1.2}$$

In Figure 1.6a we can see a total of four Hamming errors when aligning H_1 with $\tilde{H}_2$ and H_2 with $\tilde{H}_1$. This results in a Hamming rate of $\frac{4}{2m}$, because the phasing consists of $2m$ alleles. In Figure 1.6b, the two predicted haplotypes are recombined in the middle of the chromosome. This results in a total of m Hamming errors (and an HR of $\frac{1}{2}$), because we either have to count the left halves of both predicted haplotypes as errors or the right halves. This is an extreme case, where a single switch event in the phasing results in a worst-case phasing regarding the HR metric. Since switches are a common type of error in phasings, we will see this in many experiments. We call such switch events *critical switches*.



(a) Four allele errors

(b) One switch event, counted as two switch errors

(c) Two switch events, counted as five switch errors

(d) Visualization of N50 block size

Figure 1.6: **Visualization of evaluation metrics.** The examples (a), (b), and (c) visualize several types of errors between true and predicted haplotypes. (d) shows a graphical explanation of the N50 block size metric.

Switch error rate

A well-established evaluation metric for diploid phasing is the *switch error rate (SER)*, for which we introduce a polyploid version. Instead of counting the number of incorrect alleles on each haplotype, the SER counts the minimum number of switches, i.e., how often the assignment between predicted and true haplotypes must be changed in order to reconstruct the true haplotypes from the predicted ones. The polyploid extension of the switch error was already introduced as the *vector error rate* in [60].

To compute the SER, we assign a (possibly different) permutation on the predicted haplotypes for every variant position, such that the permuted predicted haplotypes match the true ones. For the SER we seek a sequence of permutations, such that the total number of changes between each two consecutive permutations is minimized. More formally, for every variant position j let $\Pi_j \subseteq S_p$ be the set of one-to-one mappings between $\mathcal{H}$ and $\tilde{\mathcal{H}}$, such that for each $\pi \in \Pi_j$ it holds that $H_i[j] = \tilde{H}_{\pi(i)}[j]$ for all haplotypes H_i. The number of switch errors is then defined as:

$$\mathrm{SE}(\mathcal{H}, \tilde{\mathcal{H}}) = \min_{(\pi_1,\ldots,\pi_m)\in\Pi_1\times\ldots\times\Pi_m} \frac{1}{p} \sum_{i=1}^{m-1} d_S(\pi_i, \pi_{i+1}) \tag{1.3}$$

where m is again the number of variants and $d_S(\pi_i, \pi_{i+1})$ is the number of different mappings between π_i and π_{i+1} – similar to the Hamming distance d_H. The SER is normalized by the factor $\frac{1}{(m-1)}$ because there are $m-1$ consecutive pairs of permutations:

$$\mathrm{SER}(\mathcal{H}, \tilde{\mathcal{H}}) = \frac{1}{m-1}\mathrm{SE}(\mathcal{H}, \tilde{\mathcal{H}}). \tag{1.4}$$

If the genotypes of $\tilde{\mathcal{H}}$ are not equal to the genotypes of $\mathcal{H}$ for all positions, then the set $\Pi_1 \times \ldots \times \Pi_m$ is empty and the vector error cannot be computed. Therefore we compare only those positions, on which the predicted genotype is correct. To account for genotype accuracy, we measure the *wrong genotype rate (WGR)*. It is defined as the fraction of variants, where $\tilde{\mathcal{H}}$ induces a different genotype than $\mathcal{H}$. The implementation of all evaluation metrics in WHATSHAP only considers variants that are heterozygous on both true and predicted haplotypes. Therefore, a heterozygous variant that is wrongly phased as homozygous does not count into the WGR. As compensation, we provide the absolute number of phased variants to reveal dropouts in the variant count.

The example in Figure 1.6b resulted in a very high HR but can be resolved with just two switches or a SER of $\frac{2}{2\cdot(m-1)}$. Figure 1.6c shows the reason why we count the number of involved haplotypes in every switch event, instead of just counting the number of such events: The first switch event affects two haplotypes, while the second event affects all three. The SER metric accounts for this and penalizes the first event as a "$\frac{2}{3}$-switch" only.

Switch flip rate

As an alternative to the SER, one could additionally allow single allele changes, called *flips*, as corrections to $\tilde{\mathcal{H}}$. This covers the case of divergent genotypes, as we can always spend (up to) p flips to transform any present genotype in $\tilde{\mathcal{H}}$ into any target genotype in $\mathcal{H}$. Flips and switches are incomparable, leaving us with a bi-objective metric in the most general case. We decided to weigh both types of errors equally and call the minimum number of combined flips and switches in relation to the total number of phased variants as the *switch flip rate (SFR)*. It can be expressed as finding the best genotype-concordant set of haplotypes $\mathcal{H}' = (H'_1, \ldots, H'_p)$, such that the number of (i) Hamming errors between $\mathcal{H}$ and $\mathcal{H}'$, and (ii) the number of switches between $\mathcal{H}'$ and $\tilde{\mathcal{H}}$ is minimized:

$$\mathrm{SFR} = \frac{1}{p \cdot m} \min_{\mathcal{H}' \in \mathscr{H}} \left(\mathrm{HE}(\mathcal{H}, \mathcal{H}') + \mathrm{SE}(\mathcal{H}', \tilde{\mathcal{H}})\right), \tag{1.5}$$

where $\mathscr{H}$ is the set of genotype-concordant haplotype sets for $\mathcal{H}$. The difference between SER and SFR can be seen in Figure 1.6a. While the SER omits the two errors on the left and right due to a genotype mismatch between the phasings, the SFR counts these as two flips. Additionally, the error in the middle can be resolved with two flips for the SFR, while the SER needs a total of four switches. This results in an SER of $\frac{4}{4\cdot(m-3)}$ and an SFR of $\frac{4}{4\cdot m}$.

N50 block size

Phasing tools may not phase the entire input region as one set of haplotypes. If the phasing between two consecutive variants is too uncertain (e.g., if not enough reads cover both variants),

the phasing might be split into *blocks*. In our evaluation, we applied the HR, SER, and SFR on all reported phasing blocks separately and aggregated them. More precisely, we summed up the number of respective errors and divided them by the total number of variants (HR, WGR) or by the total number of variants excluding the first variant in every block (SER, SFR). Since this favors shorter blocks, we also included the N50 block length in our evaluation, which is the smallest block length needed to cover 50% of the considered genomic region when using only blocks of that size and larger. Figure 1.6d visualizes this by sorting all phasing blocks by size in descending order and taking the length of the first block that passes 50% of the chromosome size. Please note, that the N50 block size is calculated on base pair count, not on the variant counts of the blocks.

1.6 Input and output

Following the theoretical framework, we will now discuss how the input and output are represented in a practical implementation. There are three relevant file formats for our developed software, which are described in detail below. We also explain the purpose of these file formats and in which steps they are used or produced.

FASTA and FASTQ files

The FASTA format is used to store plain sequence data. It is segmented into blocks that consist of a description line and an arbitrary number of data lines [61]. The description line always starts with a “>”-symbol and gives a unique identifier for the block. The data lines may contain either nucleic or amino acid sequences following the IUB/IUPAC encoding [62]. Haplotype phasing is usually done on a nucleotide level, which only covers the four bases `A`, `C`, `G`, and `T`, and `N` for an unknown base. The reference sequence for the target species is stored in a FASTA file.

As mentioned in Section 1.1, the first step in haplotyping is to generate sequencing data from an individual in the form of reads. These reads could be stored in a FASTA file as well, but in practice, the FASTQ format is the standard option [61, 63]. It is organized in blocks as well, with a description line starting with “@”, followed by several data lines containing raw sequences. In addition, there is a separator line (consisting of just a “+”-symbol) after the data lines, followed by position-wise Phred-scaled quality scores. The purpose is to provide an estimate of measuring error for each base in a sequence.

SAM and BAM files

SAM files contain alignment records for a set of reads, consisting of an alignment position (inside the reference sequence), a so-called CIGAR string to describe the alignment, and auxiliary information like alignment quality, alignment type, strand, and more [61]. These files are produced by alignment tools like BWA-MEM [31] or Minimap2 [32], using an input FASTQ file with sequencing data and a FASTA file containing a reference genome. Each read can have

```
@HD VN:1.6  SO:coordinate
@SQ SN:chr1 LN:300
@PG ID:samtools PN:samtools VN:1.16.1   CL:samtools view -h -b example.cuts.sam
@PG ID:samtools.1   PN:samtools PP:samtools VN:1.16.1   CL:samtools view -h example.cuts.bam
Read00  0   chr1    91  60  8M  *   0   0   TTATTATT    F77BBFFF
Read01  0   chr1    101 60  8M  *   0   0   TTATTATT    FFBFFDDB
Read02  0   chr1    101 60  5M1X2M  *   0   0   TTATTCTT    FFBFBBFF
Read03  0   chr1    101 60  8X  *   0   0   GGCGGCGG    BBBF77FF
Read04  0   chr1    101 60  1M1X1M1X1M1X1M  *   0   0   TGAGTCT FBBFFII
Read05  0   chr1    101 60  1M1X1M1X1M1X1M  *   0   0   TGAGTCT F79BBFF
Read06  0   chr1    101 60  1M1X1M1X1M1X1M  *   0   0   TGAGTGT FBFB99F
```

Figure 1.7: This example shows the header and some alignment records of a SAM file. The header defines a single chromosome `chr1` with length 300 and lists executed commands to generate this file. Alignment records contain (from left to right) a read name, alignment flags (e.g. primary or secondary alignment, strand, etc.), chromosome name, position inside the chromosome, alignment score, CIGAR string, three values used for linked or paired-end reads, the read sequence and the base qualities.

multiple alignment records, one of which is tagged as primary alignment. This is either used to provide multiple alignment positions (if the read could not be mapped to a unique reference location) or to track a fragmented alignment (if a read contained long indels and could thus not be aligned in one piece).

The beginning of the file contains several header lines, starting with "@" to provide additional information, e.g. what chromosomes exist in the alignment file or what *read groups* are present. Each alignment can be assigned to one read group, which allows to store reads from multiple sources (e.g. multiple individuals) in one file. Figure 1.7 contains a small example to visualize the basic fields of a SAM file. The bitwise alignment flag allows to store various binary properties per alignment. In our case, it is always 0 which stands for a primary alignment in forward direction. BAM files are binarized SAM files, which are compressed in BGZF format.

VCF files

VCF files store information about genetic variation of one or multiple individuals. The variant records contain a chromosome identifier, a position (inside the referenced chromosome), a reference allele, at least one alternative allele, and some optional fields, which are defined in the header of the file [64]. The most important optional field for our purpose is the genotype field ("GT"), which is used by convention to store the genotype information for each variant and for each individual. The genotype is a list of integers, separated by "/" or "|", where each integer stands for an allele index as explained in Section 1.2.2. The "/"-symbol is used for unphased genotypes, which by convention are sorted in ascending order (for example, 1/2/0 would be written as 0/1/2). Phased variants are indicated by the "|"-separated alleles, inducing an order on the alleles. Each haplotype $\tilde{H}_i$ can be read by always taking the i-th entry in each "|"-separated field. Homozygous variants are always stored with "/"-symbols because there is nothing to phase.

A reference-based phasing algorithm takes a VCF file as input, as it defines which variants and which alleles exist. The present unphased genotypes are then to be converted into

```
##fileformat=VCFv4.1
##FILTER=<ID=PASS,Description="All filters passed">
##contig=<ID=chr01>
##INFO=<ID=CIGAR,Number=A,Type=String,Description="The extended CIGAR representation of each alternate allele.">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=PS,Number=1,Type=Integer,Description="Phase set identifier">
##FORMAT=<ID=AD,Number=R,Type=Integer,Description="Allelic depths for the ref and alt alleles in the order listed">
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO    FORMAT  NAMETBD
chr01   17  .   A   C   100 PASS    CIGAR=1X    GT:PS:AD    1|0|1|0:17:45,37
chr01   45  .   C   CGG 100 PASS    CIGAR=2I    GT:AD   0/0/0/0:85,1
chr01   72  .   G   A   100 PASS    CIGAR=1X    GT:PS:AD    1|0|0|1:17:41,42
chr01   89  .   G   A,C 100 PASS    CIGAR=1X    GT:PS:AD    0|0|1|2:89:46,18,19
chr01   90  .   A   C   100 PASS    CIGAR=1X    GT:PS:AD    0|1|1|0:89:43,40
```

Figure 1.8: This example shows the header of a VCF file and five variant records. All but the second variant are phased. The four phased variants are distributed over two phasings blocks with IDs “17” and “89”. The second variant is an insertion of two G-bases, the other variants are SNPs, with the fourth one being multi-allelic. Allele depths are given as a comma-separated list of integers.

phased genotypes, representing the output of the algorithm. Cut positions are usually expressed through a *phase set* field (“PS”) that assigns an identifier to each phased genotype. All genotypes with the same identifier are assumed to form one coherent phasing block. For WHATSHAP, the PS field contains the position of the first variant inside a phasing block as an identifier. In Chapter 4 we will also use *allele depth* information from the VCF file. That is, the corresponding AD field contains a comma-separated list of absolute frequencies on how often each of the defined alleles has been observed inside a read set of each individual. This information can be derived from a BAM file as well but is sometimes directly stored inside the corresponding VCF file for convenience. An example for all the described fields is presented in Figure 1.8. VCF files are created by variant callers for which we already mentioned example tools in Section 1.2.2. Variant calling is a separate computational problem and will not be covered in this thesis.

Allele matrices

To our knowledge, there is no standardized file format to represent allele matrices that are required for our theoretical framework. Thus, the translation of read alignments into such allele matrices is handled by the phasing tool itself. This includes the search of spanned variants for each read based on the records of both the VCF file and the BAM file. While this process is straightforward for SNPs, it can become tricky and ambiguous for insertions and deletions. Since all our presented methods are implemented as part of the WHATSHAP software, they use the same allele detection routine as the originally published tool. In short, WHATSHAP can use a provided reference genome to re-align each input read to (i) the reference sequence itself and (ii) to a modified sequence where a single reference allele is replaced by an alternative. For each read, the best-scoring allele is determined for each spanned variant. If no reference genome is provided, each read’s alleles are estimated from the CIGAR string. Since this thesis does not contribute to the existing detection routine, we refer to [12] for further reading.

Chapter 2

Trio-phasing on human data

The first method we want to present in this thesis is designed for diploid phasing problems. Diploid phasing is the most commonly-used application for haplotyping because humans are diploid organisms and human genetics is – due to its many medical applications – likely the largest field of research in computational genomics. Thus, diploid phasing is far better explored than polyploid phasing with a variety of solving algorithms having been proposed over the years.

In this chapter, we will review one of these methods, namely WHATSHAP, a widely-used and maintained tool that was first presented in 2015 by Patterson et al. [12] for phasing single individuals and later extended by Garg et al. [13] to process related individuals. One drawback of the used model is the poor scaling with sequencing depth as the runtime of the algorithm grows exponentially with the highest read coverage within the input data. In practice, the algorithm has been shown to yield good results by downsampling the input data to a feasible read coverage. We want to investigate whether the procedure can be improved by keeping all the input data but instead not solving the model to optimality.

We implemented a heuristic for both the single-sample and related-sample methods and compared its performance to the existing implementation on sequencing data from a real human sample for which a gold-standard assembly is available. Although our implementation is available in WHATSHAP since version v2.2, it is still considered experimental because there have been no tests on real-world data prior to this thesis.

2.1 Diploid phasing

Diploid phasing is a special case of haplotype phasing that allows for a few simplifications compared to the more general polyploid phasing. We will give a short introduction to diploid phasing in general, the widely-used MEC model, and the generalized PedMEC model, which extends the MEC model to multiple related individuals.

2.1.1 Properties of diploid genomes

In Section 1.2.2, we introduced genotypes to express the abundance of different alleles for each variant position. In haplotype phasing, homozygous variants are usually omitted as the haplotype sequences are identical at these positions, and thus there is nothing to compute. Only if one does not have genotype information or does not trust them, homozygous variants are considered for phasing. In this thesis, however, we will always omit variants that have been marked as homozygous.

In addition, we assume all variants to be bi-allelic, i.e., for each variant there only exist two eligible alleles with indices 0 and 1, respectively. We justify the second assumption with the fact that multiple different mutations from the reference genome at the same position are very rare, given the overall low mutation density on human genomes; SNPs occur about once every 1000bp as the most frequent variant type. Therefore, only three different genotype patterns – `0/0`, `0/1`, and `1/1` – exist for our diploid scenario, out of which two are homozygous.

If we restrict computed phasings to follow the given genotypes, the two computed haplotypes will always be complementary to each other: For every variant, exactly one haplotype contains the reference allele, while the other one contains the alternative allele. Thus, it is sufficient to only know one haplotype as the other can be easily derived by replacing each allele a from the first haplotype with $1-a$. The assumption of both haplotypes being complementary is commonly used among phasing tools and is called the *all-heterozygous assumption*.

2.1.2 The (w)MEC model

There are a couple of computational models to solve the diploid phasing problem. We will focus on the *Minimum Error Correction (MEC)* model that is widely used among diploid and even some polyploid phasing algorithms. Based on the idea that every read originates from either of the two haplotypes and contains randomly sampled sequencing errors, the MEC model seeks to compute a bipartition of all reads in a parsimonious way.

For a formal definition, let $\mathcal{A}$ be an allele matrix with n rows and m columns. A row R_i of $\mathcal{A}$ (representing a read) *conforms* with a haplotype $\tilde{H}$ if either $R_i[j] = \tilde{H}[j]$ or $R_i[j] = -$ for every $j \in \{1, \ldots, m\}$. An allele matrix $\mathcal{A}$ is *conflict-free* if there exist two haplotypes $\tilde{H_0}$ and $\tilde{H_1}$ such that every row in $\mathcal{A}$ either conforms with $\tilde{H_0}$ or $\tilde{H_1}$. For the all-heterozygous assumption, $\tilde{H_0}$ and $\tilde{H_1}$ additionally need to be complementary, i.e. $\tilde{H_1}[j] \neq \tilde{H_2}[j]$ for every $j \in \{1, \ldots, m\}$. We state the weighted form of MEC – also called wMEC – as the following optimization problem:

Problem 3 (wMEC). *Given an allele matrix $\mathcal{A}$ and a non-negative weight matrix $\mathcal{W} \in \mathbb{R}^{n \times m}$, $W_{ij} \geq 0$, find a set F of entries such that $\mathcal{A}$ becomes conflict-free if we switch all entries in F from 0 to 1 and vice versa. Over all such sets, we want to find the set F with minimum cost, i.e. $\sum_{(i,j) \in F} \mathcal{W}_{i,j}$ is minimized.*

We call each switch-operation induced by the set F a *flip*. Another way to characterize the wMEC problem is a partitioning problem, where the goal is to find a bipartition of the rows

of $\mathcal{A}$ such that all rows within a partition are conform to one another (in the same manner as reads conform to haplotypes) using a minimum-weighted set of flips. The weight matrix $\mathcal{W}$ can be used to express the confidence for each called allele, i.e., how well a read was mapped against the reference genome for a certain locus. For the unweighted version of wMEC – just called MEC –, one can set all weights of W to 1 to effectively count the number of flips.

The MEC model was introduced by Lippert et al. [65] among other theoretical models to infer haplotypes. Cilibrasi et al. [66] later showed that the MEC problem is NP-hard, even without weights and read gaps, i.e., gaps may only occur at the beginning or end of each read but not between two defined alleles. In the following, we will use reads and rows interchangeably; the same holds for variants and columns.

Related work

In the context of haplotype phasing, several algorithms for solving the MEC model have been proposed. For the sake of highlighting different approaches we will list a selection of algorithms, as the focus of this chapter lies on our developed heuristic of the WHATSHAP algorithm. One of the earliest exact solvers of the MEC model in terms of haplotype phasing is a branch-and-bound approach by R.-S. Wang et al. [67]. This method, however, only scaled to a small number of columns and the authors additionally proposed a heuristic for larger instances. In 2008, V. Bansal and Bafna presented the tool HapCUT, which transforms the allele matrix into a graph and minimizes the MEC score by iteratively computing max-cuts on the derived graph [68]. Although it outperformed other heuristics and yielded good results in practice, it is not guaranteed to find the optimal solution. He et al. used dynamic programming to provide another exact solver [69]. The benefit was a runtime of $\mathcal{O}(n \cdot m \cdot 2^k)$, where k is the length of the longest read, i.e., the range between the first and last non-gap position. In contrast to previous exact approaches, it scales well with large allele matrices but is only applicable to short-read technologies. Other approaches that were published afterwards use integer linear programming to find optimal solutions for given MEC instances [70, 71]. This works well if the sequencing data is divided into smaller disconnected sub-matrices beforehand such that these sub-matrices can be solved independently without losing global optimality. However, some of these sub-matrices proved to hard to solve within a reasonable time limit, since solving integer linear programs is an NP-complete problem itself.

2.1.3 The PedMEC model

In this section I summarize and present a previously existing phasing model introduced in [13].

The mentioned separation of the allele matrix into independent sub-matrices also points to a major limitation occurring in haplotype phasing: A contiguous phasing of an individuals chromosome is only possible if all variants are connected by the input reads. With read length

being limited by the sequencing technology and the long homozyguous regions being present on the human genome, this prerequisite is rarely given, independent from the used phasing model.

In Section 1.3 we already introduced pedigree-based phasing as a means to use genetic information from related individuals to overcome the limitations of pure read-based phasing. Garg et al. proposed an extension of MEC – called PedMEC – to process multiple sets of sequencing data simultaneously [13]. The related individuals are given as a set of indices $\mathcal{I}$ with trio relationships $\mathcal{T} \subseteq \mathcal{I}^3$ to model the mother-father-child relations. Every individual may occur at most once as a child among all trios and $\mathcal{T}$ must form a proper family tree without circular relationships. Instead of having a single allele matrix $\mathcal{A}$, we are now given one matrix $\mathcal{A}_i$ for each individual $i \in \mathcal{I}$. All matrices share the same set of variants, and thus contain m columns. This means that we consider all variants for our phasing problem, for which at least one individual is heterozygous. In reverse, the all-heterozygous assumption does not hold anymore for every individual. The goal is to compute one pair of haplotypes $\tilde{H}_0^i$, $\tilde{H}_1^i$ for each individual i, such that the MEC model is optimized accordingly.

At this point, we have $|\mathcal{I}|$ independent instances of MEC, which we could solve separately. Thus, we introduce additional constraints between these instances that are derived from the biological reproduction process we described in Section 1.2.3. Let $(i_m, i_f, i_c) \in \mathcal{T}$ be a trio from family $\mathcal{I}$. The child i_c possesses one maternal haplotype (say $\tilde{H}_0^{i_c}$) and one parental haplotype (say $\tilde{H}_1^{i_c}$). The maternal haplotype is a mosaic of two haplotypes $\tilde{H}_0^{i_m}$ and $\tilde{H}_1^{i_m}$ from i_m, i.e., for every variant position $1 \leq j \leq m$ either $\tilde{H}_0^{i_c}[j] = \tilde{H}_0^{i_m}[j]$ or $\tilde{H}_0^{i_c}[j] = \tilde{H}_1^{i_m}[j]$ or both must hold. The same holds for the paternal haplotype of c and the two haplotypes of father f. We define two *transmission vectors* $t_{i_m \to i_c}, t_{i_f \to i_c} \in \{0,1\}^m$ to describe which haplotypes of the parents were passed to the child for each position j: The haplotypes $\tilde{H}_0^{i_m}, \tilde{H}_1^{i_m}, \tilde{H}_0^{i_f}, \tilde{H}_1^{i_f}, \tilde{H}_0^{i_c}, \tilde{H}_1^{i_c}$ are *compatible* with $t_{i_m \to i_c}$ and $t_{i_f \to i_c}$, if

$$\tilde{H}_0^{i_c}[j] = \tilde{H}^{i_m}_{t_{i_m \to i_c}[j]}[j] \qquad \wedge \qquad \tilde{H}_1^{i_c}[j] = \tilde{H}^{i_f}_{t_{i_f \to i_c}[j]}[j] \tag{2.1}$$

for every variant j. In other words, if $t_{i_p \to i_c}[j] = 0$, child i_c inherits the allele from the first haplotype of parent i_p, and otherwise the allele from the second haplotype.

Every time a transmission vector contains different values for two consecutive variants j and $j+1$, our solution contains a *recombination event*. Since these are only very few recombination events per chromosome per individual, we have to restrict their use via some penalty in our model. Let $\mathcal{X} \in \mathbb{R}^{m-1}, \mathcal{X}[j] \geq 0$ be a vector of non-negative *recombination cost*, which is provided as additional input. Every recombination event between variants j and $j+1$ on any of the transmission vectors adds a cost of $\mathcal{X}[j]$ to the model. Garg et al. formulate the additional constraints and objectives as the PedMEC model:

Problem 4 (**PedMEC**). *Given a family of individuals $\mathcal{I}$, a set of trio relationships $\mathcal{T}$, a vector of recombination costs $\mathcal{X}$ and an allele matrix $\mathcal{A}_i$ for every $i \in \mathcal{I}$, find (i) sets F_i of flips for every $\mathcal{A}_i$ such that $\mathcal{A}_i$ becomes conflict-free yielding haplotypes $\tilde{H}_0^i, \tilde{H}_1^i$ and (ii) a pair of transmission*

vectors $t_{i_m \to i_c}, t_{i_f \to i_c} \in \{0,1\}^m$ *for every* $(i_m, i_f, i_c) \in \mathcal{T}$ *such that* $\tilde{H}_0^{i_m}, \tilde{H}_1^{i_m}, \tilde{H}_0^{i_f}, \tilde{H}_1^{i_f}, \tilde{H}_0^{i_c}, \tilde{H}_1^{i_c}$ *are compatible with them. The objective is to minimize the following function:*

$$\sum_{i \in \mathcal{I}} |F_i| + \sum_{(i_m, i_f, i_c) \in \mathcal{T}} \sum_{j=1}^{m-1} \mathcal{X}[j] \cdot \left(\sum_{i_p \in \{i_m, i_f\}} [\![t_{i_p \to i_c}[j] \neq t_{i_p \to i_c}[j+1]]\!] \right) \tag{2.2}$$

For the sake of simplicity, we stated the unweighted version, which can be generalized to weighted flips like the wMEC model. For a full description of the weighted problem and more formal details, we refer to the original publication.

2.2 Algorithm of WHATSHAP

I summarized and rephrased the explanations from [12] for the exact algorithm. The heuristic is a novel work and has not been presented before in any publication.

In 2015, Patterson et al. presented WHATSHAP, a fixed-parameter-tractable approach to optimally solve the wMEC problem depending on only one parameter, the maximum coverage among the reads. Thus, the runtime grows exponentially with the maximum coverage on any variant position. However, in contrast to previous methods, the approach scales linearly in the number of variants and does not depend on the length of the longest read, enabling full-chromosome phasing on long reads.

In practice, WHATSHAP performs a read selection on the input allele matrix to crop the maximum coverage to some parameter c while maintaining as much connectivity information as possible. In this section, we will introduce a heuristic for the exact algorithm that is able to process much higher coverages. For that purpose, we will briefly summarize the existing algorithm using an adapted notation while omitting some implementation details.

2.2.1 Solving MEC with dynamic programming

The basis of WHATSHAP is a dynamic program (DP) that computes optimal partial solutions for subsets of the columns in $\mathcal{A}$. It considers all possible bipartitions of rows in $\mathcal{A}$, which would be 2^n candidate solutions without any optimization. The heuristic solves the same dynamic program but only memorizes a small portion of the partial solutions when advancing to the next variant, which effectively loses the guarantee for optimality.

Exact algorithm

We borrow some of the notation introduced by Patterson et al. and adapt it to our existing framework; for every variant v_j, let $A_j \subseteq \{1, \ldots, n\}$ be the set of row indices i, such that $s(R_i) \leq j \leq e(R_i)$. We call A_j the *active set* for variant (or column) j because all associated reads

are either defined on v_j itself or possess a non-gap allele before and after v_j. Furthermore, let $N_j := A_j \setminus A_{j-1}$ (with $A_0 = \emptyset$) be the set of *new* reads that start at column j.

A *bipartition* of a set A is a pair (P,Q) with $P \cup Q = A$ and $P \cap Q = \emptyset$. We say that (P',Q') *extends* (P,Q) if $P \subseteq P'$ and $Q \subseteq Q'$. Let $\mathscr{B}(A)$ be the set of all bipartitions over the set A and – for a given partition (P,Q) over set A – let $\mathscr{B}(A' \mid (P,Q))$ be the set of all bipartitions over A' that extend (P,Q). Finally, let $D(B,j)$ be the optimal cost for rendering the first j columns of $\mathcal{A}$ conflict-free such that the resulting bipartition extends B.

We interpret D as a large table with a total of m columns – one for each variant – and one row for every considered bipartition $B \in \mathscr{B}(A_j)$ in each of the columns. The individual entries $D(B,j)$ represent partial solutions and the principle of the dynamic program is to compute them bottom-up, i.e., using solutions for small sub-problems to derive solutions for larger sub-problems. In their work, Patterson et al. describe how D can be efficiently computed in total time $\mathcal{O}(2^c \cdot m)$, where c is the maximum coverage, i.e., $c = \max_{1 \le j \le m} |A_j|$. The idea is to compute D column-wise by considering all bipartitions from $\mathscr{B}(A_1)$ for the first column and then extending bipartitions from one column j by all reads N_{j+1} starting at column $j+1$. The optimal global bipartition can then be computed by starting at the minimum score in the last column $D(\cdot, m)$ and using backtracking pointers to combine all local bipartitions on the optimal path into a global one. For the rest of this section, we will use the running example illustrated in Figure 2.1. The columns of D for the example allele matrix and the associated bipartitions can be seen below the labels $\mathscr{B}(A_1)$, $\mathscr{B}(A_2)$, and $\mathscr{B}(A_3)$.

For every column j, the algorithm only memorizes $D(B,j)$ for every $B \in \mathscr{B}(A_j)$ instead of every $B \in \mathscr{B}(\{1,\ldots,n\})$. The reason is that a read R_i with $s(i) > j$ does not induce any conflicts in the first j columns of $\mathcal{A}$ and does not need to be tracked for the first j columns of D. Each read only enters the scope of the algorithm at its first defined position $s(i)$. Likewise, a read R_i with $e(i) < j$ leaves the scope of the algorithm for column j because it does not contribute anymore to the scores $D(B,j)$.

For their DP recursion, Patterson et al. introduced *intermediate projection columns* $D^{\cap}$, defined as

$$D^{\cap}(B,j) = \min_{B' \in \mathscr{B}(A_j \mid B)} D(B',j) \tag{2.3}$$

for all $B \in \mathscr{B}(A_j \cap A_{j+1})$. It collapses all bipartitions from $\mathscr{B}(A_j)$ that only differ in reads ending at column j into a single intermediate solution. When computing column $j+1$, every intermediate bipartition $B \in \mathscr{B}(A_j \cap A_{j+1})$ is effectively expanded into $2^{N_{j+1}}$ new ones that represent all possible extensions of B using new active reads from A_{j+1} (that were not active before).

This collapse-and-expand procedure is illustrated in Figure 2.1 for a small example with four reads and three variants. For the first column, we see all possible bipartitions $\mathscr{B}(A_1)$ over A_1, out of which two have a score of 0 and the other two have a score of 1. Since the first read moves out of scope for the second column, all bipartitions that only differ in that read are

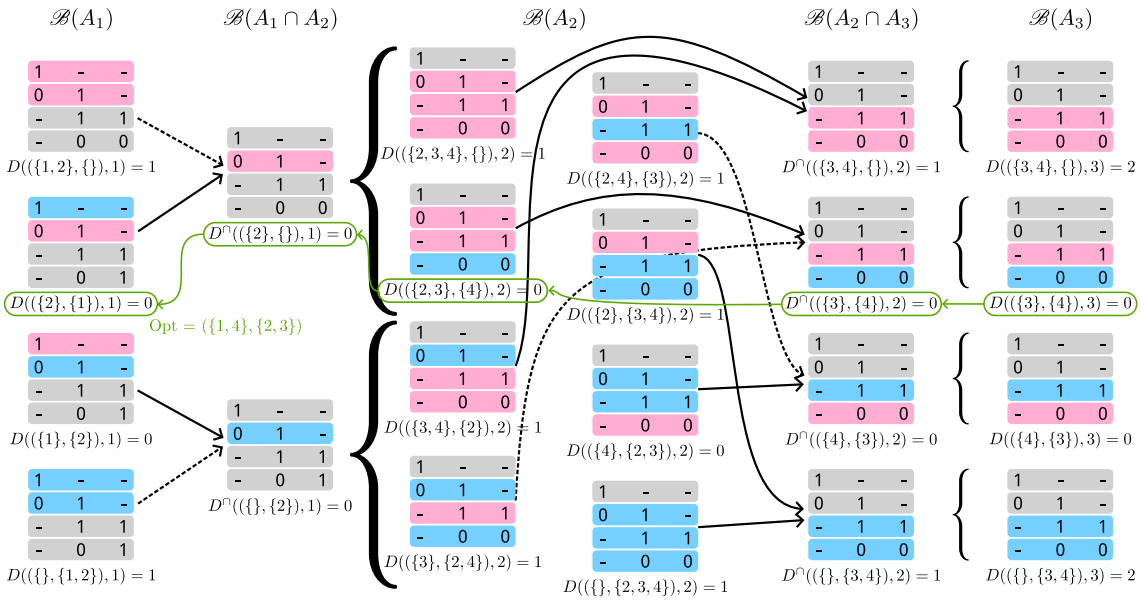


Figure 2.1: **Example for the WHATSHAP algorithm.** This overview shows the principle of the WHATSHAP algorithm on a small example with four reads and three variants. Local bipartitions (or solutions) are represented by small matrices with their rows colored in magenta for the first partition, blue for the second partition, and gray for inactive reads. For each bipartition, the score below each matrix shows the D-values for the main columns and the auxiliary $D^{\cap}$-scores for intermediate projection columns. The black arrows show which bipartitions from the main columns are considered for the auxiliary scores, where dashed lines stand for suboptimal predecessors. In reverse, curly brackets show which bipartition from an intermediate projection column is extended to which bipartitions in the main column. The optimal solution and the backtracking over all columns are highlighted in green.

collapsed. The second column $\mathscr{B}(A_2)$ introduces two new reads and the algorithm extends each bipartition from the intermediate column into four new bipartitions and computes their respective scores. Once all columns are processed, the bipartition with the lowest score (or one of them if multiple exist) in that column is picked and the resulting global bipartition is constructed by following the backtracking pointers and inserting each read into the corresponding partition.

Heuristic

The major limitation of the dynamic program is the exponential scaling with read coverage: For variant j, the DP has to keep track of all $2^{|A_j|}$ bipartitions over these reads. This even holds if some reads contain gaps at variant position j because a read counts as active between its first and last non-gap position. To maintain practical runtimes, the implementation of WHATSHAP performs a read selection to only keep a subset of reads such that there are no more than K active reads at any variant for some user-defined threshold K. The default threshold is set to 15, while values above 20 quickly become impractical.

In this section, we propose a heuristic for the original algorithm that aims to limit the number of memorized bipartitions at any point in time to some threshold L, and thus avoid combinatorial explosion in areas of high coverage. Since we cannot ensure that the optimal solution will always be preserved by the selection of L bipartitions, this approach represents a heuristic for the underlying MEC problem with no performance guarantees. Our heuristic is

inspired by the polyploid phasing tool H-PoP-G by M. Xie et al. [40]. Their algorithm solves a generalized version of the MEC problem for polyploid genomes but only does so heuristically due to the huge solution space. In the following, we will use the terms bipartition and solution interchangeably because every bipartition B in column j can be interpreted as a partial solution with a MEC score of $D(B, j)$.

The first major difference between our heuristic and the exact algorithm is how columns are handled; the latter enumerates all bipartitions $\mathscr{B}(A_j)$ over the active reads A_j, finds the (optimal) predecessor from $\mathscr{B}(A_{j-1} \cap A_j)$ – we called this intermediate projection column – , and computes the induced flip cost for the new column j. Every bipartition from $\mathscr{B}(A_{j-1} \cap A_j)$ is extended by $|N_j|$ new elements (i.e. the number of reads newly starting at column j), resulting in $2^{|N_j|}$ possible extensions.

For the heuristic, it is impractical to generate all $2^{|N_j|}$ extensions for the intermediate projection column because $2^{|N_j|}$ can potentially be a large number itself. In real-world sequencing data, one can observe that there are positions where many read alignments start simultaneously. This can, for example, be caused by a large insertion in the sequenced individual. Since the insertion is not part of the reference genome, the affected parts of some reads cannot be aligned and their alignments start at the first position that is present in the reference again. Therefore, it is a realistic scenario that N_j can be too large for some columns j to enumerate all possible bipartitions over it.

We tackle this problem by processing all columns read-wise: We duplicate the previous set of bipartitions and insert the new read into the first or second partition, respectively. Afterward, we order the resulting bipartitions by their score $D(B, j)$ and discard the worst elements until at most L bipartitions are left. To reduce random choices inside our heuristic, we always discard all equally good solutions as a whole, which might result in less than L remaining solutions if the L-th and $(L + 1)$-th best solutions (and possibly more) have the same score. However, we always keep all best solutions, even if there are more than L of them. We also copy the concept of intermediate projection columns from the exact algorithm to collapse solutions that only differ in reads that become inactive in the next column.

We reuse the input matrix from Figure 2.1 to illustrate the principle of our heuristic: Figure 2.2 shows all performed steps for a solution limit of $L = 2$. We initially start with an empty bipartition and a score of 0. In the first column, the first read can be either inserted into the first (magenta) or second (blue) partition, yielding two different solutions with a score of 0. Both solutions can be extended with the second read by inserting it into either partition. The algorithm keeps the best $L = 2$ solutions and prunes the rest. With all reads of the first column being processed, we compute the intermediate projection column by collapsing bipartitions that only differ in the first read. Since we discarded two out of four solutions, we do not collapse multiple bipartitions and instead just keep the two with the first read now marked as inactive. In the second column, we extend the two kept solutions into four solutions by inserting the third read into either partition. However, the algorithm now keeps all four solutions – despite the limit of $L = 2$ – because all solutions are equally good. After processing the fourth

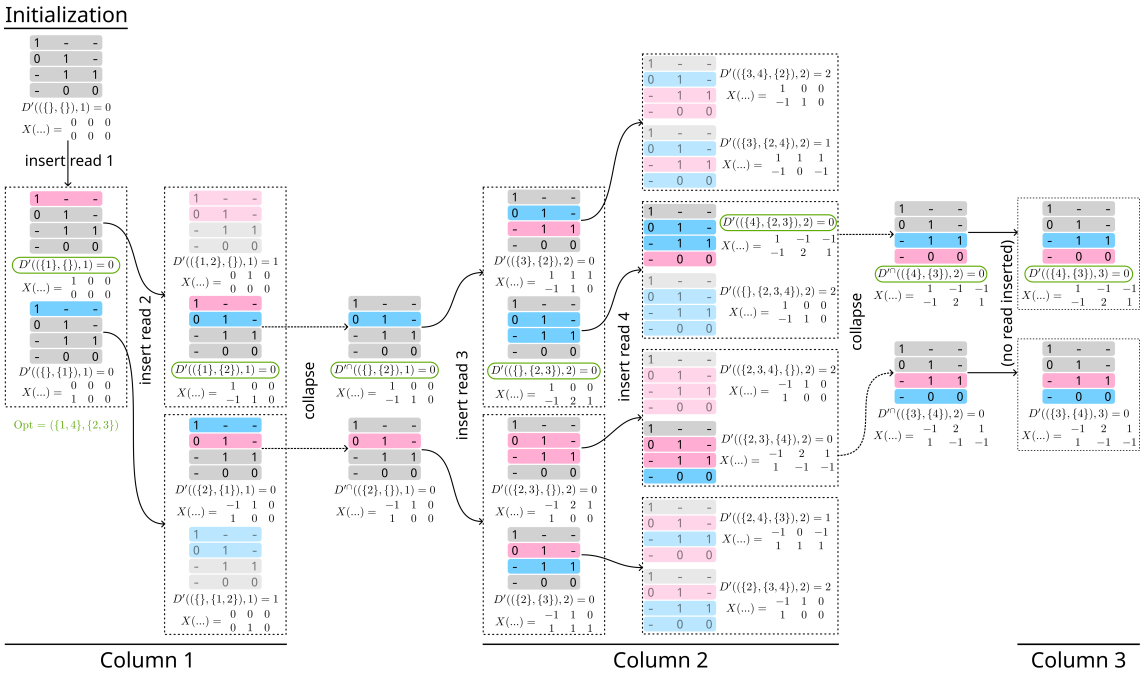


Figure 2.2: **Example for the wMEC heuristic.** This overview shows the heuristic for the WHATSHAP algorithm. Bipartitions are represented as in Figure 2.1. The algorithm starts with an empty bipartition and a score of 0. Black arrows indicate the extension of a previous bipartition into two new ones that are framed by a dashed box. Solutions that are discarded by the algorithm due to the solution limit $L = 2$ are depicted with lower opacity. Dashed arrows indicate a collapsing step for the intermediate projection columns. In the third column, there is no extension, hence only one bipartition inside each of the dashed boxes. The bipartitions on the optimal path are highlighted with a green border around the score. The X-values indicate allele balances for each solution with the top row belonging to the magenta partition and the bottom row to the blue one. The arguments for X are the same as for D' but are not depicted for better visual clarity.

read, we can finally discard six out of eight solutions. The second intermediate projection column does not involve any collapsing and only copies the two remaining solutions with the second read marked as inactive. Since no read starts in the third column, there is nothing to do and the algorithm constructs an optimal solution via backtracking from the lowest score in the last column in the same manner as the original WHATSHAP algorithm. Note that the X-values in Figure 2.2 indicate the allele balances for each intermediate solution; we will introduce these values throughout this section and refer back to this figure.

During development, we realized that pruning solutions based on the D-values is not effective: If for some bipartition B two reads R_i and R_k reside in the same partition, they also will for all extensions of B and all following columns. The score $D(B, j)$, however, only considers the alleles of R_i and R_k up to column j. If R_i and R_k have no conflicts before and up to column j but many conflicts thereafter, bipartitions having R_i and R_k in the same partition are preferred, although a globally good solution would separate them. Figure 2.3 shows an example where the D-scores could lead to a bad pruning choice: Inserting the fourth read into the blue partition (solution B_1) yields a better D-score on the second column than inserting it into the magenta one (solution B_2). Thus, B_2 would be pruned before B_1. Among all extensions of B_1 and B_2, however, the best solution is found when extending B_2 instead of B_1. In practice, such cases are realistic because a read starting in column j might contain a sequencing error on its

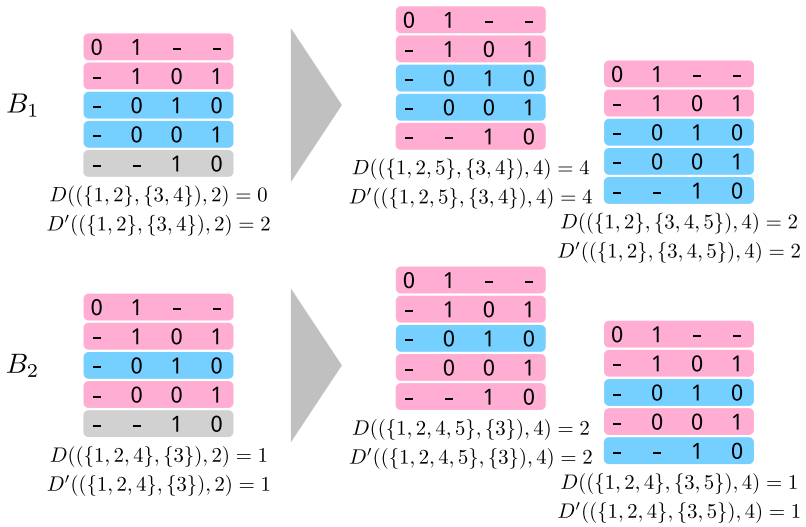


Figure 2.3: **Difference between DP scoring schemes.** On the left side, there are two bipartitions B_1, B_2 for the first four reads and their scores for Column 2 according to D and D'. B_1 and B_2 differ in their assignment of the fourth read; this results in B_1 to achieve a better score for the D-scheme than B_2, while the opposite holds for D'-scheme. On the right side, we can see the two extensions of B_1 and B_2 (by including the last read) and their scores for Column 4. Both D and D' are identical for of the four final bipartitions.

first non-gap position, and could thus be sorted into the wrong partition based on this single erroneous allele.

To circumvent this issue, we introduce a slightly different DP score $D'(B, j)$ that indicates the optimal cost for rendering the rows $P \cup Q \cup \bigcup_{k=1}^{j-1} A_k$ conflict-free such that the resulting bipartition extends $B := (P, Q)$. In other words, the score considers all active reads from the first $j-1$ columns and those reads from A_j that are already assigned a partition by B. The difference to the definition by Patterson et al. is that our score is not strictly limited to the first j columns but rather includes rows of $\mathcal{A}$ as a whole once they move into the scope of the algorithm.

Note that $D'(B, m) = D(B, m)$ for all $B \in \mathcal{B}(A_m)$; thus, computing a DP matrix with our score definition would lead to the same solution as the algorithm by Patterson et al. but with a different calculation for each entry and possibly different results for each but the last column. We emphasize that the purpose of the D'-scheme is not to compute a different solution than the original one but to provide a better indicator which intermediate solutions to discard. We found that our scheme yields vastly better results than the D-scheme, especially for low values of L. Therefore, we will omit the former scheme and only discuss the latter from now on. Strictly speaking, our heuristic actually solves a slightly different DP than WHATSHAP but since the overall MEC model and the solving principle remain the same, we still call it a heuristic for the WHATSHAP algorithm.

To efficiently compute $D'(B, j)$, we introduce an auxiliary table X with $X(B, j)$ containing the *allele balances* for the (intermediate) solution associated with $D'(B, j)$. Each entry consists of two so-called *balance vectors* of length m, one for each partition. The k-th component indicates a single *allele balance*, i.e., the number of reads with 1-alleles minus the number of reads with 0-alleles in the respective partition for column k. For the weighted version, we

add and subtract allele weights instead of counting the number of allele occurrences. We denote balance vectors as $X(B,j)_q$ for $q \in \{1,2\}$ and single allele balances as $X(B,j)_q[k]$ for $k \in \{1,\ldots,m\}$. For the example in Figure 2.2, the corresponding allele balances are given below each D'-entry. The initial empty solution starts with allele balances that consist only of 0s for both partitions. Every time a solution $B = (P,Q)$ is extended by read R_i, the balance vector for the extended partition – say P – is updated according to the read's alleles (and their weights), which can be done in time $\mathcal{O}(e(R_i) - s(R_i))$.

The update of the D'-scores is simpler without the all-heterozygous assumption because we only have to consider the updated balance vector. Incrementing a positive allele balance or decrementing a negative allele balance means that R_i follows the present consensus in P and does not induce additional flip costs for the affected column. In reverse, we have to raise the D'-score each time $R_i[k]$ deviates from the consensus of the other reads in P for all variants $s(R_i) \leq k \leq e(R_i)$. If the weight $\mathcal{W}_{i,k}$ for read R_i at column k is lower than the absolute allele balance $|X(B,j)_1[k]|$ of P for column k, the consensus changes by inserting R_i and we need to pay $\mathcal{W}_{i,k}$ flipping cost, i.e., increment the current D'-entry by $\mathcal{W}_{i,k}$. Otherwise, the consensus changes due to the insertion of R_i in P and we only pay costs $|X(B,j)_1[k]|$. Formally, the updated score can be computed with

$$D'((P \cup \{R_i\},Q),j) = D'(B,j) + \sum_{k=s(R_i)}^{e(R_i)} \min\left(\mathcal{W}_{i,k}, |X(B)_1[k]|\right) \cdot [\![d(R_i[k], X(B)_1[k])]\!], \quad (2.4)$$

where $B = (P,Q)$, R_i is inserted into the first partition P, and $d(a,x)$ is true if and only if allele $a \in \{0,1\}$ does not follow the consensus implied by the allele balance x, i.e.,

$$d(a,x) = (a = 1 \wedge x < 0) \vee (a = 0 \wedge x > 0)\,. \quad (2.5)$$

If both haplotypes have to be complementary, we have to consider both balance vectors of $X(B,j)$ because an allele of the inserted read R_i can cause costs, even if it conforms with the consensus of the target partition P. This happens if both partitions have a positive balance for column k alongside $\mathcal{W}_{i,k}$ but P possesses the lower balance value and would be selected to be the 0-allele to satisfy the all-heterozygous assumption. Analogously, if both partitions have a negative score with P possessing the greater balance, the latter would be assigned to carry the 1-allele. Thus, a read induces additional flip cost for column k if $R_i[k] = 0$ and $X(B,j)_1[k] > X(B,j)_2[k]$ or if $R_i[k] = 1$ and $X(B,j)_1[k] < X(B,j)_2[k]$, i.e., the effective consensus after the all-heterozygous assumption does not match the read allele. Using $\bar{X}(B,j,k)$ as a short notation for $X(B,j)_2[k] - X(B,j)_1[k]$, Equation (2.6) summarizes the score computation based on balance vectors for this assumption:

$$D'((P\cup\{R_i\},Q),j) = D'(B,j) + \sum_{k=s(R_i)}^{e(R_i)} \begin{cases} \min\left(\mathcal{W}_{i,k}, \max\left(0, \bar{X}(B,j,k)\right)\right) & \text{if } R_i[k]=1, \\ \min\left(\mathcal{W}_{i,k}, \max\left(0, -\bar{X}(B,j,k)\right)\right) & \text{otherwise.} \end{cases} \tag{2.6}$$

Note that the exact algorithm would not profit from the more complex definition of D' over D because it enumerates all possible solutions for each column anyway, and thus never has to rank and select partial solutions.

2.2.2 Extending MEC solvers to PedMEC

The exact algorithm for PedMEC is existing work by Garg et al. [13] and restated here as a basis for our newly developed heuristic. All aspects of the heuristic and its documentation here are novel.

In Section 2.1.3, we introduced the PedMEC model defined by Garg et al. [13]. In the same publication, the authors also extend the existing WHATSHAP algorithm to process multiple individuals simultaneously and solve the PedMEC problem to optimality. The extension follows the same DP approach as the original algorithm, where we consider all possible bipartitions over the set of active reads A_j for every column j. The first key difference is that the reads are now scattered over multiple input allele matrices $\mathcal{A}_1,\ldots,\mathcal{A}_{|\mathcal{I}|}$. Since each read still needs to be assigned to either the first or second haplotype within its corresponding individual, we can characterize an assignment for all reads as a single bipartition. Note that we treat the reads from $\mathcal{A}_1,\ldots,\mathcal{A}_{|\mathcal{I}|}$ as a combined set of reads. Thus, the sets $A_1,\ldots,A_m$ contain the sets of active reads from *all* input matrices combined. The second key difference is that we now also consider the *transmissions* of all trios, i.e., which haplotype from each of the parents is passed on to the respective child. A single transmission is characterized by a binary vector of length $2\cdot|\mathcal{T}|$, where each pair of binary numbers stands for a single trio and indicates which haplotype – 0 for the first and 1 for the second – the child inherited from mother and father, respectively.

The authors extended the DP table by a third dimension to cover the present transmission. More formally, each DP entry $D(B,t,j)$ represents an optimal solution according to the PedMEC model to render the first j columns in all input allele matrices conflict-free, such that the solution extends the bipartition B and uses transmission t for column j. While we considered bipartitions as (partial) solutions in Section 2.2.1 and used the two terms interchangeably, we now characterize a solution as a pair of one bipartition and one transmission. We still consider the last index j of each DP entry $D(B,t,j)$ to be the column index of a solution but the row index is now determined by $(B,t)\in\mathcal{B}(A_j)\times\{0,1\}^{2|\mathcal{T}|}$, i.e., one of the possible combinations of bipartitions over A_j and transmissions for the $|\mathcal{T}|$ trios. This increases the total size of the DP table to $\mathcal{O}\left(m\cdot 2^{c+2|\mathcal{T}|}\right)$, where $c := \max_{1\le j\le m}|A_j|$ is again the maximum coverage over all columns. The global optimal solution can be retrieved via backtracking from a lowest-score

entry in the last column. The total asymptotic runtime of the exact algorithm is reported as $\mathcal{O}\left(m \cdot 2^{c+2|\mathcal{T}|+|\mathcal{I}|} \cdot |\mathcal{I}| + 2^{4|\mathcal{T}|+c}\right)$.

The DP recursion works similar to the MEC version: First, we use intermediate projection columns $D^{\cap}$ to collapse solutions that only differ in reads ending at the previous column j. Analogously to Equation (2.3), these columns are defined as

$$D^{\cap}(B,t,j) = \min_{B' \in \mathscr{B}(A_j|B)} D(B',t,j) \tag{2.7}$$

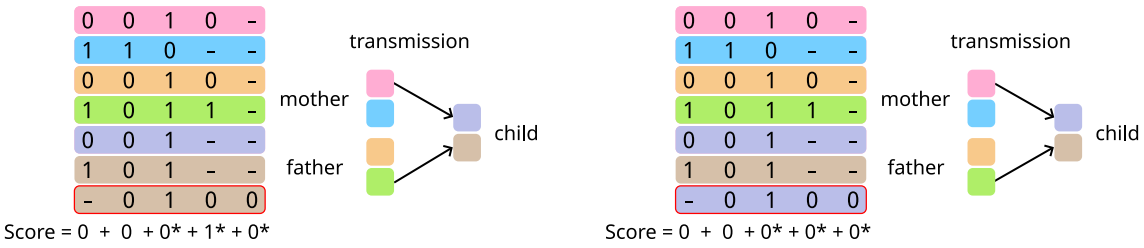
for all $B \in \mathscr{B}(A_j \cap A_{j+1})$. The second step of the recursion is to compute all extensions of the remaining solutions using the read indices N_{j+1} (i.e. all reads starting in column $j+1$). In addition to the MEC algorithm, we need to account for possible changes in transmission. When computing entry $D(B,t,j+1)$, we do not only consider all entries $D^{\cap}(B',t,j)$ with B extending B' as predecessors, but also entries $D^{\cap}(B',t',j)$ with a different transmission $t' \neq t$. For the latter, we consider the corresponding recombination cost $\mathcal{X}[j] \cdot d_H(t,t')$ with $d_H(t,t')$ being the Hamming distance between the t and t'. For further formal details on the exact PedMEC algorithm, we refer the reader to the original publication and instead focus on our developed heuristic.

Heuristic

The exact PedMEC algorithm possesses the same exponential runtime scaling as its MEC version regarding read coverage. However, this coverage does not refer to each individual separately but to all individuals at once because the sets of active reads are shared between all individuals. Thus, the average coverage per individual is downsampled to $\frac{c}{|\mathcal{I}|}$ with c being the global coverage threshold for each variant position. This only yields a coverage of 5× per individual (and a haploid coverage of 2.5×) for a single trio and the default coverage threshold of 15. Garg et al. showed that the benefits of the heredity information outweigh the loss in coverage per individual by a great margin but the high loss of information still raises the question to what extend the phasings could profit from the large amount of discarded reads.

To answer this question, we will extend our MEC heuristic, which is able to handle higher coverages, to also solve the PedMEC problem. Similarly to the exact algorithm, we consider combinations of bipartitions and transmissions as (intermediate) solutions and proceed column-wise through the given input matrices. We again use a solution limit L as an upper bound for how many solutions we memorize throughout the computation. Since solutions now also contain a transmission, it is possible that we memorize the same bipartition multiple times but with different transmissions. This implies that the PedMEC heuristic might require higher values of L for proper results compared to the MEC heuristic. We will investigate this in the experimental section.

For the heuristic itself, we borrow the new definition of the intermediate projection columns, which was introduced by the exact PedMEC algorithm. This includes the adjusted DP recursion that considers predecessors with different transmissions by adding the corresponding recom-



(a) Read insertion with required flips

(b) Read insertion without required flips

Figure 2.4: **Look-ahead scoring for transmission.** The shown matrices represent combined input allele matrices for a trio of mother, father and child. The first six rows show a bipartition with the magenta, orange, and violet rows being associated with the first haplotypes (of mother, father, and child respectively) and the blue, green, and brown rows being associated with the second haplotypes. The transmission is shown to the right of the matrix. The next read (highlighted with a red border) belongs to the child individual. **(a)** The next read is inserted into the brown partition. It does not contradict the present consensus of this partition on any position, and thus does not induce any flip costs directly. According to the given transmission, however, the brown haplotype inherits its alleles from the father's green haplotype. Since the green and brown haplotypes would become different in the fourth column after the read's insertion, we would have to pay a flip cost of 1 in the future to change either of the two haplotype's allele in the affected column and restore the transmission constraint. **(b)** The next is inserted into the violet partition. While this decision does not induce direct flip costs either, it avoids conflicts with the transmission because alleles from the violet haplotype agree with those from the mother's magenta one.

bination costs (see the explanations of Equation (2.7)). In addition to the recursion, the exact PedMEC algorithm also modified how solutions are extended in each column: When processing a new column, the algorithm enumerates all possible bipartitions over active reads and all possible transmissions. Our MEC heuristic avoids a full enumeration of bipartitions by inserting new reads one at a time and only keeping the L best solutions after each insertion until all new reads of the new column are processed. To account for the increased solution space of the PedMEC model we need two adjustments to our heuristic: (i) a mechanism to generate alternative transmissions and (ii) adjusted criteria to rank solutions for the solution limit L.

The first addition is necessary because the extension step of the MEC heuristic only extends existing bipartitions but not the transmissions. We proceed similarly to the exact algorithm: After all new reads of a column j – possibly none – are processed, we take the bipartitions of all up to L remaining solutions and create one copy of them for every possible transmission. This yields up to $L \cdot 2^{2|\mathcal{T}|}$ solutions, from which we again select the best L. This ensures that possible recombination events between columns $j-1$ and j are covered by some of the solutions. We also change the initialization of the heuristic (see Figure 2.2) to not use the empty bipartition as the only start solution but rather one copy of the empty bipartition for each of the $2^{2|\mathcal{T}|}$ possible transmissions.

The second addition aids in efficiently identifying solutions whose transmissions will lead to conflicts induced by the inheritance constraints of the PedMEC model. For the MEC heuristic, we use a different definition for our DP table compared to the exact algorithm. This allows us to use all alleles of a newly inserted read for the solution ranking and not only all alleles up to the current column j. When we add inheritance constraints to the model, just using the values from the DP table D' might overlook certain errors induced by a read insertion. In Figure 2.4 we demonstrate this issue for a small example with one trio (mother, father, and child). All reads from A_1 are assigned a partition and the next read belonging to the child

is to be inserted into either of the two partitions. Note that we used six different colors to differentiate between individuals and their two haplotypes. Subfigures 2.4a and 2.4b show the same combined allele matrix and the same transmission but with the new read inserted into different partitions. In both cases, the new read does not induce flip errors according to the allele balance concept because it has no conflicts with either the previous violet or the previous brown reads. However, if we take the transmission into consideration, putting the new read into the brown partition will potentially violate the inheritance rule between the green and brown haplotypes in the fourth column. We would then need to flip either the alleles of the green or the brown haplotype to restore a proper inheritance. It is still not an *actual cost* we already have to pay because we do not know yet what the set of solutions will be once we advance to the fourth column. It rather is a hint to potential costs or *look-ahead costs* that we could use for solution ranking. Inserting the new read into the violet partition does not induce look-ahead costs because the violet haplotype inherits from the magenta one, whose consensus is concordant with the new read. Since the look-ahead costs have to be recalculated for every column due to possible changes in transmissions, we limit the scope of the computation to the following five columns.

Homozygosity

In the introduction of this chapter, we explained the all-heterozygous assumption and that it usually is reasonable for diploid phasing. When phasing multiple individuals simultaneously we say that a variant is heterozygous if *any* of the phased individuals is heterozygous. This means that the individual allele matrices may contain columns where the corresponding individual is homozygous, even when using the all-heterozygous assumption.

Homozygous columns in one of the allele matrices become problematic if the latter contains reads that only cover such homozygous columns, which we call *homozygous reads*. Since the two haplotypes are identical on these positions, both read partitions should have the same consensus for the associated columns. Thus, a homozygous read should fit equally well into both partitions. According to how our heuristic works, a homozygous read effectively duplicates the previous set of (up to) L solutions by inserting the homozygous read once into the first and once into the second partition of each of the previous solutions. When shrinking the new solution set to at most L solutions, the duplicates will either both be taken or both be discarded because they will have the same D'-score (they might still differ in the look-ahead score in some cases). This means that the new solution set only contains $\frac{L}{2}$ different solutions if we do not count the homozygous read that does not contribute any information to the phasing. If the next reads are homozygous as well, the solution diversity is further decreased. Figure 2.5 shows an example of an allele matrix with three homozygous columns and three homozygous reads. It shows four different partitions that are equally good and only differ in the assignments of the homozygous reads.

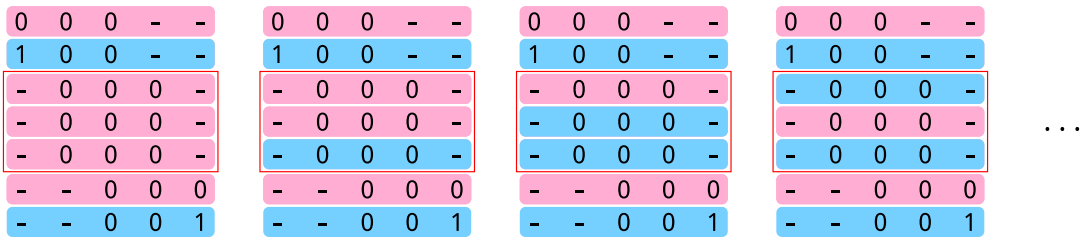


Figure 2.5: **Duplicated PedMEC solutions.** The allele matrix in this figure depicts the (error-free) reads of an individual, where the three variants in the middle are homozygous on the 0-allele. Thus, the third, forth, and fifth read (highlighted by the red box) only cover homozygous positions. The four copies of the matrix show different bipartitions of the readset (indicated by magenta or blue background), although only the three highlighted reads are shuffled between partitions. Since they only represent homozygous variants, the resulting haplotypes are identical for all eight (or 2^3) possible bipartitions.

During our experiments, we encountered the worst case several times, where more than $\log_2 L$ homozygous reads occurred in very close proximity. Not only did this reduce the effective solution diversity to 1, it even prevented the heuristic from terminating in time because it always keeps all optimal intermediate solutions. Since homozygous reads lead to a duplication of solutions, the number of equally good solutions grows exponentially with the number of homozygous reads until other reads diversify the solution set again. The exact algorithm does not suffer from this problem because it enumerates all bipartitions anyway and makes sure that this enumeration is always feasible by limiting the maximum coverage.

We added a detection for homozygous reads to prevent the solution duplication: If we fully trust the provided genotype information from the VCF file, we know in advance which columns are homozygous for which sample and can easily identify homozygous reads. We then add the read (randomly) to either the first or second partition but do not create two new solutions. If we do not trust the provided genotypes, we iterate over all covered columns j of a new read and check (i) whether the consensus of the first and second partition differs for column j or (ii) whether the insertion of the new read would change the consensus for both partitions. If either of the two conditions holds for any covered column, we consider the read informative and create two new solutions from this read. Otherwise, we treat it as a homozygous read.

De-novo mutations

The PedMEC model demands that a child allele always matches the corresponding parent allele determined by the transmission vector. However, genetic variation not only arises from genetic recombination but also from spontaneous mutations in the child's germline, called *de-novo mutations*. As the name suggests, these mutations are exclusive to the child genome, and thus do not fit the PedMEC model. In many cases, de-novo mutations can already be detected beforehand because the genotypes of the child and its parents are incompatible to Mendel's law. Examples for incompatible patterns – also called *Mendelian conflicts* – are `0/0` for both parents and `0/1` for the child or `0/0`, `0/1` for the parents and `1/1` for the child. WHATSHAP filters all Mendelian conflicts before the phasing because they cannot be handled by the PedMEC model.

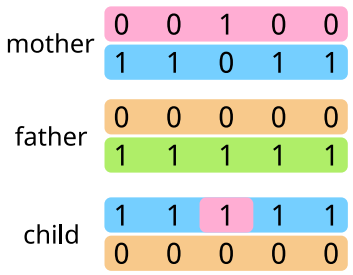


Figure 2.6: **De-novo mutation.** The colored stripes show the true haplotypes of a trio. The child contains a de-novo mutation on the third variant of the first haplotype, which does not result in a Mendelian conflict. The colors of the child haplotypes indicate the optimal transmissions according to the PedMEC model.

De-novo mutation that do not result in Mendelian conflicts will cause small artifacts in the phasing. To ensure compatibility of inherited haplotypes, the algorithm will either resolve mutations by inserting two recombination events directly before and after the affected variant or by flipping all alleles of one partition at the column of the mutation. The first case is depicted by Figure 2.6: The first child haplotype contains only 1-alleles but the closest mother haplotype contains a 0-allele on the third column. The cheapest solution could be to let the child inherit this single allele from the magenta instead of the blue haplotype, for the cost of two recombinations. Alternatively, one could report the mutated allele as 0 – the child would entirely inherit the blue haplotype – for the cost of flipping all reads in the first child partition from 1 to 0 on the third column.

To resolve mutations more elegantly, we introduced *mutation cost* $\mathcal{Y}$ as a third cost component for the phasing model and optionally allow our heuristic to insert mutations into the phasing if this leads to a reduction in total cost. Formally, we relax the inheritance constraints of the PedMEC model but add a penalty of $\mathcal{Y}[j]$ for violation of this constraint for column j. On the algorithmic side, we do not need to explicitly add mutations as events but rather change the cost calculation at the end of a column when we generate alternative transmissions for the remaining solutions. If the transmission of a solution connects two read partitions with mismatching consensus alleles (for some variant), we previously had to flip many alleles in either of the partitions (see Figure 2.4). With mutations being allowed in the phasing, we could instead pay the mutation cost once and prevent the possibly more expensive set of flips. Note that the option to allow mutations only has an effect if multiple related samples are phased; it is not applicable for single-individual phasing.

2.3 Experiments

All described experiments in this section were conducted by me and are unpublished. I took advice from my second supervisor Tobias Marschall on how to generate a proper ground-truth phasing for sample HG002.

We compared the existing implementation for the wMEC and PedMEC models with our proposed heuristic on the human Ashkenazi trio, consisting of a child (HG002) and its father (HG003) and mother (HG004). The three samples have been extensively sequenced and ana-

lyzed by the Genome in a Bottle (GIAB) Consortium [72]. Originally, we planned to reproduce the experiments from Garg et al. [13] that also use sequencing data from the Ashkenazi trio but found that the exact datasets were not available anymore. We instead used newer sequencing data, namely PacBio[1] [2] [3] reads and PacBio HiFi[4] [5] [6] reads, which were aligned to the GRCh38 reference genome[7] by Minimap2 [32].

The crucial part of the benchmark is to find a suitable ground-truth phasing to evaluate the output of the different phasers. Garg et al. ran the population-based phasing tool SHAPEIT2 [73] on a reference panel from the 1000 Genomes project [33] and unphased genotypes from the trio to construct phasings for all three samples. Since the used inputs were not available under the provided hyperlinks and all the computations were done on an older reference genome, we decided to instead use existing high-quality phasings for HG002 as ground truth. The phasings of the two parents were not evaluated.

One high-quality phasing [8] is provided by the GIAB Consortium itself [74]. It was created by combining PacBio HiFi reads and StrandSeq data using the existing PedMEC implementation of WHATSHAP. The other phasing was computed by the Telomere-to-Telomere Consortium [75]. It is available as a FASTA file with two base-level haplotype sequences[9]. In order to convert it into a phased VCF file with the same reference coordinates as the aligned reads, we used the Phased Assembly Variant Caller (PAV) [49], which was originally created for the Human Genome Structural Variation Consortium (HGSVC). A comparison between both phasings revealed a minimal deviation of less <0.03% in both SER and HR. In the following, we will use the second phasing as ground truth for our evaluation.

All metrics were measured using the `compare`-subcommand of WHATSHAP. It requires both true and computed phasing in VCF format (see Section 1.6). Since version v2.2, our heuristic is accessible through the additional parameter `-algorithm heuristic`, when running the

[1] *https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/PacBio_MtSinai_NIST/PacBio_minimap2_bam/HG002_PacBio_GRCh38.bam*

[2] *https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG003_NA24149_father/PacBio_MtSinai_NIST/PacBio_minimap2_bam/HG003_PacBio_GRCh38.bam*

[3] *https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG004_NA24143_mother/PacBio_MtSinai_NIST/PacBio_minimap2_bam/HG004_PacBio_GRCh38.bam*

[4] *https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/PacBio_CCS_15kb_20kb_chemistry2/GRCh38/HG002.SequelII.merged_15kb_20kb.pbmm2.GRCh38.haplotag.10x.bam*

[5] *https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG003_NA24149_father/PacBio_CCS_15kb_20kb_chemistry2/GRCh38/HG003.SequelII.merged_15kb_20kb.pbmm2.GRCh38.haplotag.10x.bam*

[6] *https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG004_NA24143_mother/PacBio_CCS_15kb_20kb_chemistry2/GRCh38/HG004.SequelII.merged_15kb_20kb.pbmm2.GRCh38.haplotag.10x.bam*

[7] *https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/references/GRCh38/GCA_000001405.15_GRCh38_no_alt_analysis_set.fasta.gz*

[8] *https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/HG002_NA24385_son/NISTv4.2.1/GRCh38/SupplementaryFiles/HG002_GRCh38_1_22_v4.2.1_benchmark_phased_MHCassembly_StrandSeqANDTrio.vcf.gz*

[9] *https://github.com/marbl/HG002*

phase-subcommand of WHATSHAP. However, for the following experiments we used a version from an experimental branch because we fixed a couple of smaller errors that we encountered during our evaluation. The instructions how to retrieve the applied state of the code can be found in Appendix D.

All experiments were run on an AMD Epyc 7742 64-core processor with 1 TB of RAM and Debian Kernel. They were organized as a Snakemake pipeline [76] with jobs being executed in parallel, when possible. All jobs involving the phasing tools were run on a single core to maximize comparability regarding runtime and peak memory consumption.

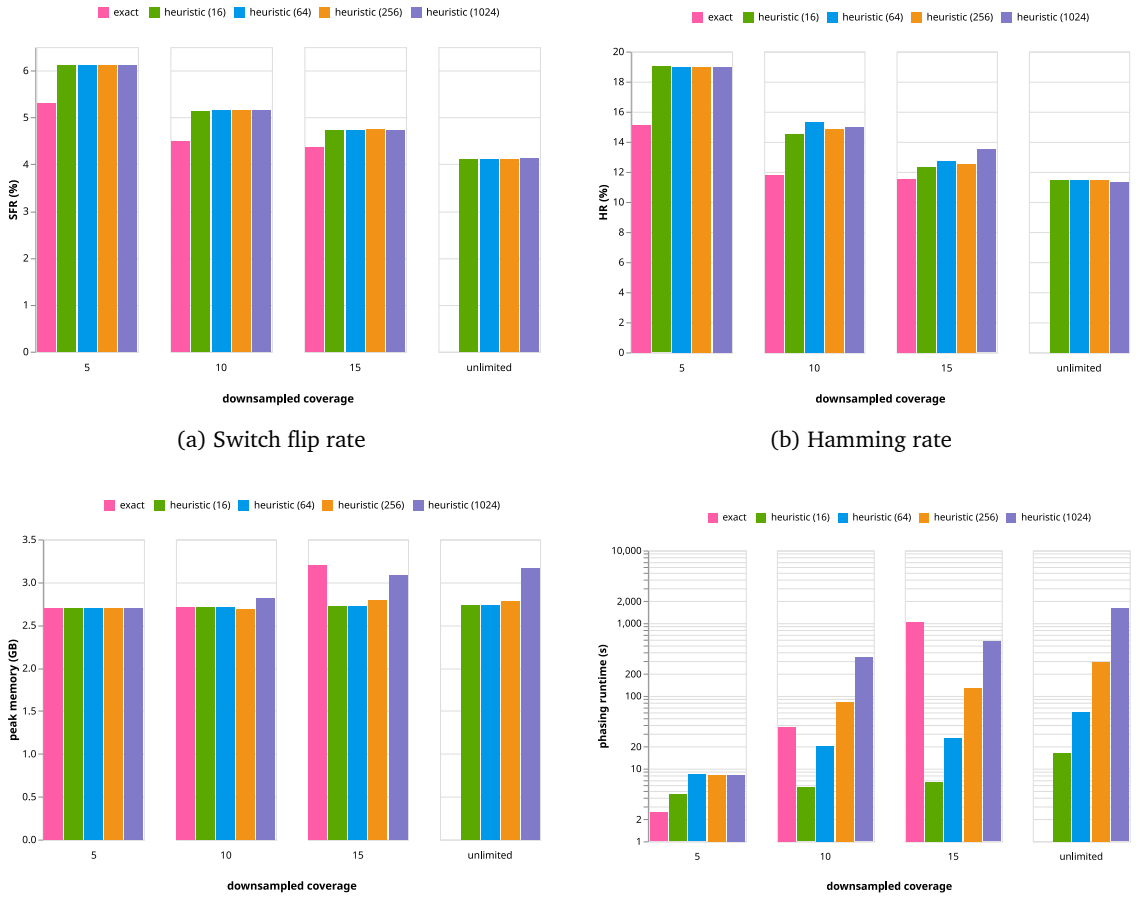
2.3.1 Single-sample phasing on HG002

The first set of benchmarks exclusively tests the single-sample phasing capabilities of the introduced heuristic. Since we only have a ground truth for sample HG002, we ignore the sequencing data for the other two samples in this section. We tested the exact algorithm of WHATSHAP for the downsampled coverages 5×, 10×, and 15× with the last being the default value. We applied the same downsampling for the heuristic to directly measure the impact of the loss of optimality. In addition, the heuristic was run on the full dataset (about 48× for the PacBio and 80× for the HiFi reads) without any downsampling. For the exact algorithm, this was not possible because of the exponential runtime scaling with coverage. As a second parameter, we varied the solution limit L for all heuristic runs. The chosen thresholds were 16, 64, 256, and 1024.

We retrieved the peak memory consumption and the runtimes from the command line output of WHATSHAP. It utilizes the integrated functions of Python to query the highest memory occupation during the process and the standard timer from the "time"-package. For the runtime measures, we only state the time spent inside the MEC solvers instead of the total runtime of the process. We decided to omit the other steps of the tool to emphasize the runtime differences between the two algorithms and the solution limit thresholds. Otherwise, the configurations with low coverage or low values for L would be dominated by the input processing steps.

In Figure 2.7, we see the results for the PacBio sequencing data: The two top plots show the switch flip rate (SFR) and Hamming rate (HR) phasing metrics and the two bottom plots indicate the required computational resources. The heuristic yields a 10–20% higher SFR than the exact algorithm among the three downsampled coverages. For the full dataset, the heuristic is slightly more accurate than the exact algorithm with coverage 15×. The solution limit only has a negligible effect on the error rate; the phasing quality is dominated by the amount of used data. The HR follows a similar pattern, although the exact algorithm is about 25% ahead for the lowest coverage of 5×. The values for L show some differences here with the tendency that higher values result in a higher HR.

For lower coverages, both results do not match our expectations: No column should contain more than 32 solutions for a coverage of 5× and the heuristic should compute the same result



(a) Switch flip rate

(b) Hamming rate

(c) Peak memory consumption

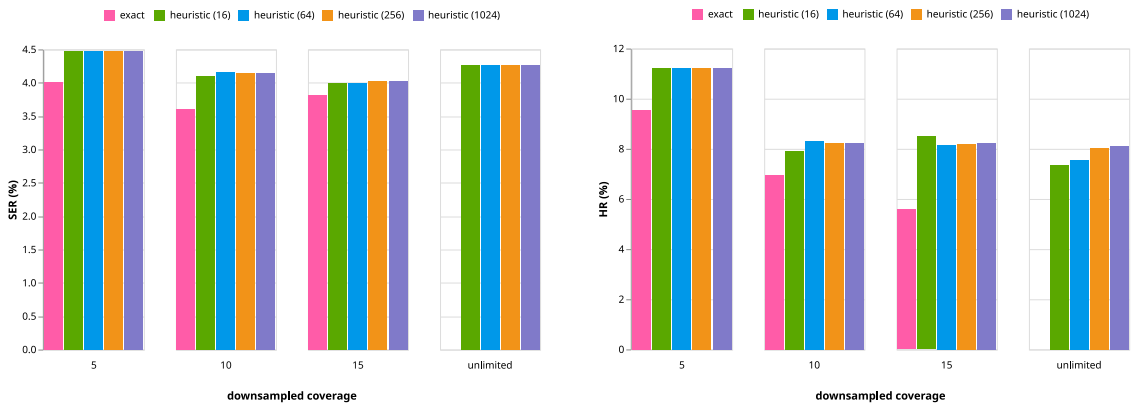
(d) Runtime (phasing algorithm only)

Figure 2.7: **Evaluation of MEC heuristic (PacBio).** Each of the bar plots compares the exact algorithm of WHATSHAP to the heuristic on the PacBio sequencing data. The shown metrics are SFR (a), HR (b), peak memory consumption (c), and runtime (d). The heuristic was run with different values of the solution limit L, which are given in parentheses behind heuristic's name. The bar plots are grouped by the used downsample threshold; the threshold "unlimited" means that the heuristic was run without downsampling. Missing bars indicate that the affected configuration was unable to finish within 24 hours or ran out of memory. The runtime only includes the phasing algorithm itself as reported by the command line output of WHATSHAP.

as the exact algorithm if $L \geq 32$. However, we noticed that our heuristic phases slightly more variants on the provided dataset for each coverage level, as shown in Table 2.1. The lower the coverage the larger the gap in phased variants and the larger the discrepancy in error rates. We did not resolve this issue in time but the data suggests that the heuristic might report some low-confidence variants (e.g. with ambiguous consensus), while the exact algorithm avoids those until it gains more evidence through higher coverage. The solution limit did not influence the number of phased variants.

algorithm	5×	10×	15×	unlimited
exact	236599	237502	239260	n/a
heuristic	240740	240909	240975	241062

Table 2.1: Number of phased variants for both algorithms, depending on the supplied coverage.



(a) Switch flip rate

(b) Hamming rate

Figure 2.8: **Evaluation of MEC heuristic (HiFi).** These plots show the SFR (a) and the HR (b) for the exact algorithm and the heuristic on the HiFi sequencing data. They follow the same format as described in Figure 2.4.

The memory consumption is very similar on all tested runs, which shows that the VCF table and the unfiltered readset (before the downsampling) dominate the memory footprints. Only for the 15× coverage on the exact algorithm and for the highest solution limit of 1024, the size of the DP tables starts to impact the memory consumption.

As expected, the exact algorithm shows an exponential runtime scaling with increasing coverage. The heuristic also requires more time for more data but the scaling is very flat compared to the exact algorithm and also compared to the solution limit L. For coverage 15×, the heuristic is always faster than the exact algorithm. However, when the heuristic memorizes the same number of solutions per column (or slightly less), e.g., for $L = 1024$ and coverage 10× or for $L = 16$ and coverage 5×, the exact algorithm is significantly faster. The reason for this discrepancy is likely that the exact algorithm uses a gray-code enumeration of all solutions in each step to minimize the computation time of the DP values. The heuristic cannot make use of such optimizations and rather focuses on sparsely storing and processing large solution spaces.

We decided to omit the switch error rate (SER) because it was closely correlated with the SFR in all experiments, and thus provided no additional insights.

For the HiFi data, we summarized the phasing quality results in Figure 2.8. In contrast to the PacBio reads, the SFR stabilizes much faster with growing coverage and at only about half of the error rate. The heuristic is not able to match the accuracy of the exact algorithm, even when using the full data set. Like for the other dataset, raising the solution limit does not improve the accuracy of the heuristic. All of this holds for the HR, except for a downsampled coverage of 5×, which results in about 35% more Hamming errors for both the exact algorithm and heuristic. Runtime and memory consumption are very similar to the PacBio dataset. The individual results can be found in Supplementary Figure A.1.

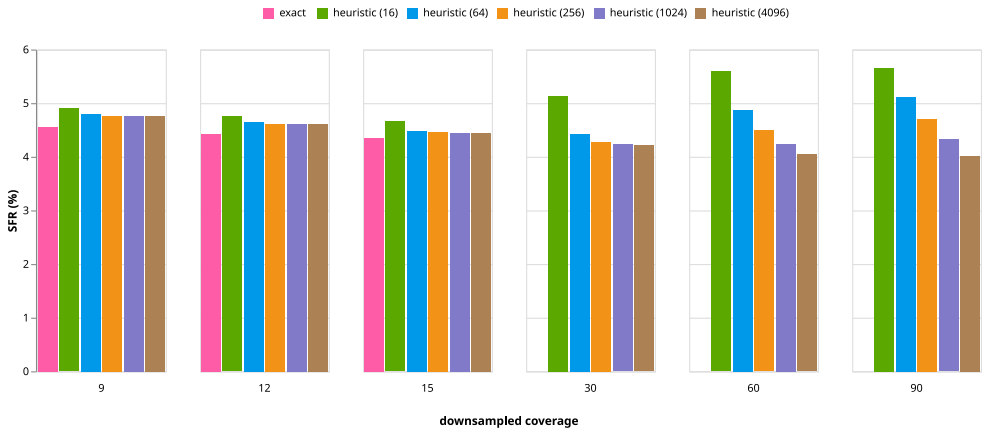
2.3.2 Trio-phasing on Ashkenazim trio

Our setup for trio-phasing is close to the setup for single-sampling phasing with a few differences: First, we use 9×, 12×, 15×, 30×, 60×, and 90× as downsampling thresholds. We extended the set of tested solution limits by $L = 4096$ because increasing the limit above 1024 proved to impact the results, as opposed to the single-sample phasing, where the solution limit parameter only had minimal effect. In the absence of a ground truth for the parental samples HG003 and HG004, all reported error metrics only refer to the phasing accuracy of the child sample HG002. The feature of allowing de-novo mutations in the phasing was still experimental at submission time, and thus disabled for most of our benchmarks. We only enabled for a set of experiments at the end of this section.

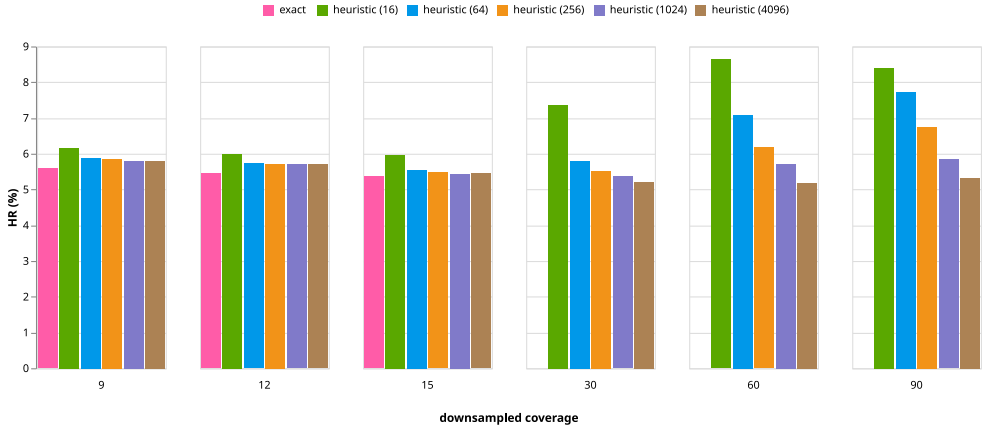
Figure 2.9 shows the SFR, the HR, and the phasing runtime for the PacBio readsets. Exact algorithm and heuristic are much closer with up to 11% for $L = 16$ and around 3–5% for $L \geq 64$. While a deeper coverage benefits the results for up to 15×, we can see some regressions when increasing the coverage further. For 30×, the lowest solution limit of 16 shows both higher SFR and HR than for 15×, while larger limits of 256 or higher see a decline in errors. Going to 60× and 90×, the larger solution limits also fall behind in accuracy, until $L = 4096$ is the only configuration to still have an advantage over downsampling the reads to 15× and to slightly beat the exact algorithm on its highest tested coverage. We keep this observation in mind for now and further investigate this in the discussion.

The runtime follows the same pattern as for the single-sample experiments with the exact algorithm scaling exponentially with coverage, while the heuristic runtime mainly scales with the used solution limit. Except for the lowest coverage setting, using $L = 4096$ is always slower than running the exact algorithm with coverage 15× by a substantial margin. Using a coverage of 30×, the default setting of $L = 256$ offers a similar performance as the exact algorithm on coverage 10× with a 30% lower runtime, while the higher parameter $L = 1024$ matches the exact algorithm on coverage 15× with a 50% lower runtime. We also planned to run the heuristic without downsampling again but the heuristic was not able to finish any of the tested configurations within the time limit of 24 hours. The reason for that are vast peaks in local coverage, on which the heuristic gets stuck if no downsampling is applied. We previously claimed an average coverage of 48× for the PacBio reads of HG002 but the median coverage is noticeably lower with just 37×. On the other end of the spectrum, the 99th percentile reaches 64× and the maximum even higher than 14,000×. In practice, it is advisable to limit the coverage to a reasonably high value, as the mentioned peaks are the result of mapping artifacts and should therefore be excluded from the phasing. Given the observed regressions, we did not expect to gain any more insights for testing coverage thresholds beyond 90× and therefore omitted further tests in this direction.

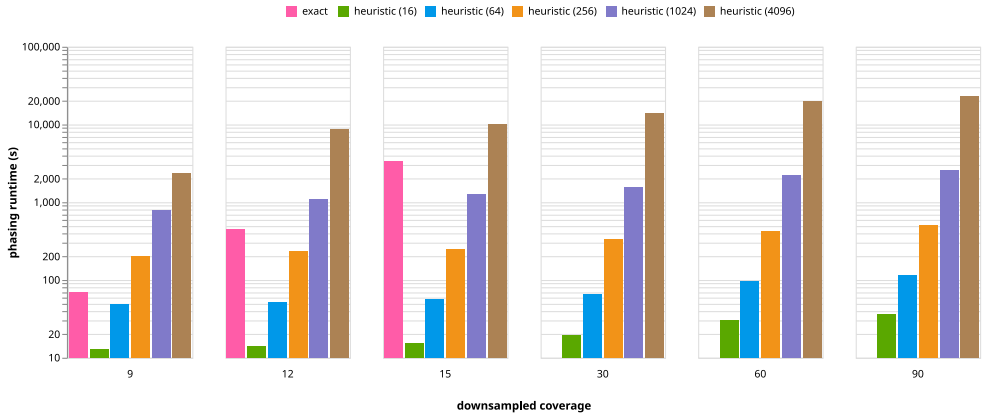
We repeated these experiments for the HiFi reads but omit a detailed description here, as the findings are the same as for the PacBio reads. The corresponding plots can be found in



(a) Switch flip rate

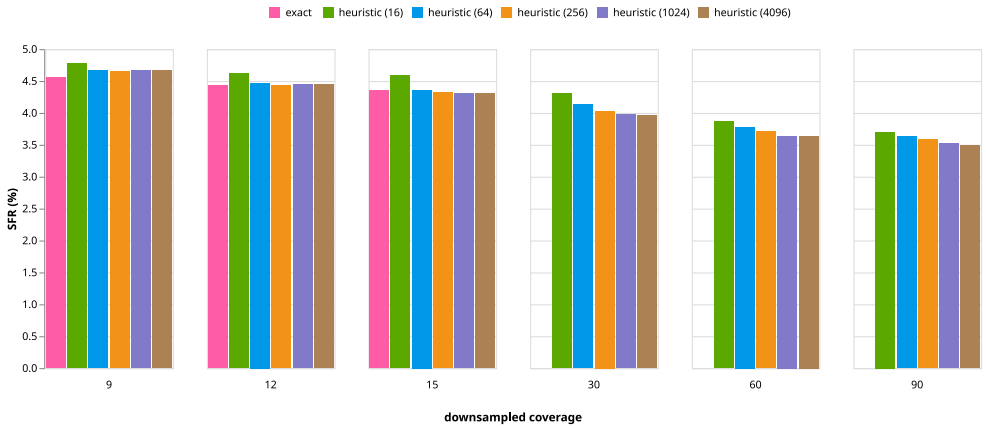


(b) Hamming rate

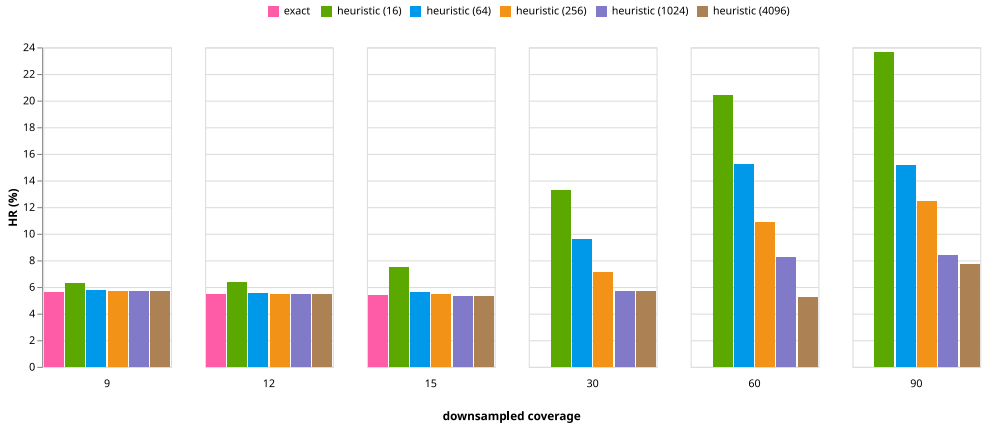


(c) Runtime (phasing algorithm only)

Figure 2.9: **Evaluation of PedMEC heuristic (PacBio).** These plots show the SFR (a), the HR (b), and the runtime (c) for the exact algorithm and the heuristic on the PacBio sequencing data. In contrast to the previous figures, all algorithms used the PedMEC model and the reads from all three samples. The plots follow the same format as described in Figure 2.4.



(a) Switch flip rate



(b) Hamming rate

Figure 2.10: **Evaluation of PedMEC heuristic with de-novo mutations.** Analogously to Figure 2.9, these plots show the SFR (a) the HR (b) for the trio-phasing on the PacBio data but the heuristic was allowed here to add de-novo mutations to its phasing.

Supplementary Figure A.3. The peak memory consumption for trio-phasing on PacBio reads is given in Supplementary Figure A.2.

In Section 2.2.2 we explained that the exact algorithm does not allow for de-novo mutations in its output, while our heuristic can optionally do so if this contributes to a lower PedMEC score. We re-ran the trio-phasing on the PacBio reads with de-novo mutation output enabled to evaluate the effect of this additional option. We see from Figure 2.10 that the SFR indeed slightly decreases and now scales much more consistently with higher coverages. The same holds for HR on coverages up to 15× but beyond that the error rates drastically shoot up and even surpass those from the test without de-novo mutations allowed. In the introductory example for the error metrics (see Figure 1.6) we pointed out that few switch errors in critical places can lead to disproportionally high Hamming rates, which might be the case here for the values above 20%. Since the detection of de-novo mutations is still a highly experimental feature, we did not investigate any further in that matter; we rather wanted to show that this feature can positively impact the phasing quality in certain cases.

2.4 Discussion

Our goal in this chapter was to present a heuristic for the exact wMEC and PedMEC solver used in the tool WHATSHAP. The heuristic should be able to process deeper read coverages, while providing near-optimal solutions and, ultimately, outperform the exact model through the use of more data. Our experiments show that the heuristic finds reasonable phasings for both single-sample and trio phasing, which are close to the optimally computed solutions in terms of SFR and HR. For the single-sample instance on the PacBio reads, the heuristic scales very well with higher coverage and slightly outperforms the exact algorithm. The heuristic is very fast for the single-sample instances because it already reaches its peak accuracy with the lowest tested solution limit of 16. For the trio-phasing instances, the solution space becomes larger and the heuristic needs benefits from 256 or 1024 memorized solutions, especially for higher coverages. The runtime is still competitive with the exact algorithm when comparing settings with equally good phasing quality.

2.4.1 Limitations and issues

However, we also discovered a couple of issues that would need to be resolved to offer consistent behavior and noticeable benefit over the existing solution. First, the heuristic does not find equally good solutions for very low coverages, even when the solution limit is sufficient to store all bipartitions for each column. When describing the functionality of our heuristic, we claimed that our DP should result in the same phasing (or one with the same wMEC score) if all solutions can be memorized. However, this is not reflected in the experiments. We already pointed out that the heuristic reproducibly reports more phased variants than the exact algorithm and therefore suffers from more errors if these additional variants cannot be phased confidently with the given data.

The second issue is the performance regression for higher coverages on the PedMEC model. While the regular wMEC model experiences an uplift in phasing quality with more provided coverage, the opposite holds true for PedMEC, unless the solution limit is set to very high values. The fact that this issue only occurred for multiple samples indicates that homozygous regions on some of the samples could be the problem here. We illustrated in Section 2.2.2 that if a read only covers homozygous variants, it can always be inserted into either of the two partitions when extending an existing solution. This results in a duplication of solutions and finally in a loss of diversity among the L memorized solutions. The higher the coverage, the more reads are enclosed in a single homozygous stretch and the more solution diversity is lost through excessive solution duplication. Increasing the solution limit compensates this problem to some degree and we can indeed observe that larger solution limits suffered less from an increase in coverage.

Another observation we should be critical of is the overall scaling with the solution size parameter. For the two single-sampled datasets, a solution limit of 16 yields the same phasing quality as a limit of 1024. This could mean that the global optimal solution is always very close

to the local optimal solution in every column and the problem can indeed be solved with such little resources. Since the heuristic is slightly behind the exact algorithm regarding the quality metrics, an alternative explanation could be an ineffective use of larger solution limits, such that the globally optimal solution is often times lost even when memorizing many solutions.

The final issue concerns the acquired test data. Although we ensured that the ground truth phasing is valid, the overall error rates of both phasing methods ended up surprisingly high. In the work of Garg et al., the exact PedMEC reached switch flip rates of less than 1% for a coverage of 15×, which is closer to our expectations – especially when using the high-quality HiFi reads. Both algorithms show consistent behavior, i.e. the quality becomes better with higher read coverages and solution limits (except for the aforementioned issues in the heuristic). We were unable to clarify where these discrepancies to tests from Garg et al. originate from.

2.4.2 Future work

The described issues regarding the experimental results of the heuristic should be resolved in future releases, for which additional testing and engineering are required. In addition to the solving capabilities of the wMEC and PedMEC models, the inclusion of de-novo mutations should receive more attention in the future. We demonstrated that the option to insert such mutations into the computed phasing can lead to an uplift in phasing quality for the PacBio reads of the tested trio. However, we did not discuss the amount of inserted mutations and how plausible it is, given that such mutations only rarely occur in reality. We should also mention that WHATSHAP filters out all variants with Mendelian conflicts beforehand, i.e. variants on which the child genotype cannot be produced from the parental genotypes through the Mendelian inheritance rules with any of the four possible transmissions. This is a useful filter, as the exact PedMEC solver cannot deal with de-novo variants. For the heuristic, these variants do not need to be filtered but could rather provide more input data. As a summary, the feature of handling de-novo mutations is still in an experimental state and could be improved and validated in a follow-up project.

Chapter 3

WhatsHap Polyphase

In this chapter, we introduce WHATSHAP POLYPHASE, an algorithm we developed to solve the polyploid phasing problem. To put it into context, we will first discuss the differences between diploid and polyploid phasing, what methods have been proposed in the past, and which caveats should be considered when dealing with polyploid genomes (Section 3.1). The main part of this chapter will explain the underlying steps and models used in our algorithm. This part is divided into three sections that reflect the structure of the algorithm: read clustering (Section 3.2), haplotype threading (Section 3.3), and refinement (Section 3.4). Finally, we will present experimental results, in which we benchmarked our new algorithm against existing methods (Section 3.5).

This work is based on [14], which was published in Genome Biology. I share first authorship with Jana Ebler and Rebecca Serra Mari. My main contribution to this work was the development of the clustering stage (presented in Section 3.2). The majority of this chapter was rewritten for this thesis and some aspects of the algorithm extended compared to the original work. Sections that re-use passages or content from [14] are marked.

3.1 Polyploid phasing

For the design of a polyploid phasing model, it is crucial to understand the differences between diploid and polyploid genomes and their implications for the phasing problem. The immediate observation is the higher number of homologous haplotypes. This greatly increases the solution space, and thus makes haplotype reconstruction considerably harder. Additionally, polyploidy raises new problems: allele dosage, sequence multiplicity (or collapsed regions), and unresolvable connections, which we further explain in the next section. Because the nature of these issues is novel compared to diploid phasing, they require further methodological engineering.

3.1.1 Caveats of polyploid in comparison to diploid phasing

This section uses ideas already presented in [14].

Allele dosage involves the presence of more possible genotypes per variant. A common assumption in diploid phasing is that all phasable variants are heterozygous since they can usually be well discriminated from homozygous ones during variant calling. It immediately follows that the genotype `0/1` is assumed by all (correctly called) phased variants. By contrast, there are several levels of heterozygosity in polyploid genomes because not only the presence of an alternative allele has to be determined, but also its *dosage*, i.e. the number of occurrences on the sequenced genome. In a tetraploid genome, for instance, the genotypes `0/0/0/1`, `0/0/1/1`, and `0/1/1/1` are all heterozygous but imply different haplotype configurations. While the detection of correct allele dosages has to be done outside the phasing routine, we have to anticipate that the input genotypes for our methods might contain dosage shifts and should not trust them to the same degree as in the diploid case.

The second issue is the occurrence of high local similarity between different haplotypes, which we call *collapsed regions*. The term describes the phenomenon of two or more haplotypes sharing (almost) the same alleles over a long interval of variants. In diploid genomes, this would simply result in a region without heterozygosity, sometimes called *variation deserts*. With more than two haplotypes being present, however, it is possible to have two (or more) duplicated sequences while still retaining heterozygosity through the unaffected haplotypes. This can be problematic for MEC-based phasing models, which seek to partition all reads into p partitions in a parsimonious way without respecting that read coverage should be evenly split among the partitions. Inside a collapsed region, reads from multiple affected haplotypes become indistinguishable and could be assigned to the same partition in the MEC model. Since every read has to be assigned to one partition, the MEC model might be inclined to use the vacant partitions over the collapsed regions to "dispose" of noisy reads that did not fit into other partitions with little correction cost. As a result, only one affected haplotype in a collapsed region is reconstructed by the shared sequence, while the other haplotype(s) could be potentially reconstructed arbitrarily. Figure 3.1 illustrates such a case for a tetraploid example, where an optimal MEC solution squashes reads from two haplotypes into a single partition. We conclude that a suitable model for polyploid phasing should encourage a uniform coverage distribution among the haplotypes to detect locally duplicated sequences and correctly assign them.

The final issue is a consequence of collapsed regions: If a collapsed region is substantially longer than the average read, it creates ambiguity when joining the phased haplotypes on the left and right sides of the region. Even if we can perfectly phase every variant outside of the collapsed region, we could be left without reads spanning the entire region, and thus be unable to uniquely identify the correct way of joining the haplotype pieces. Figure 3.2 illustrates the issue for a triploid case. The magenta and blue sequences are identical on the four central

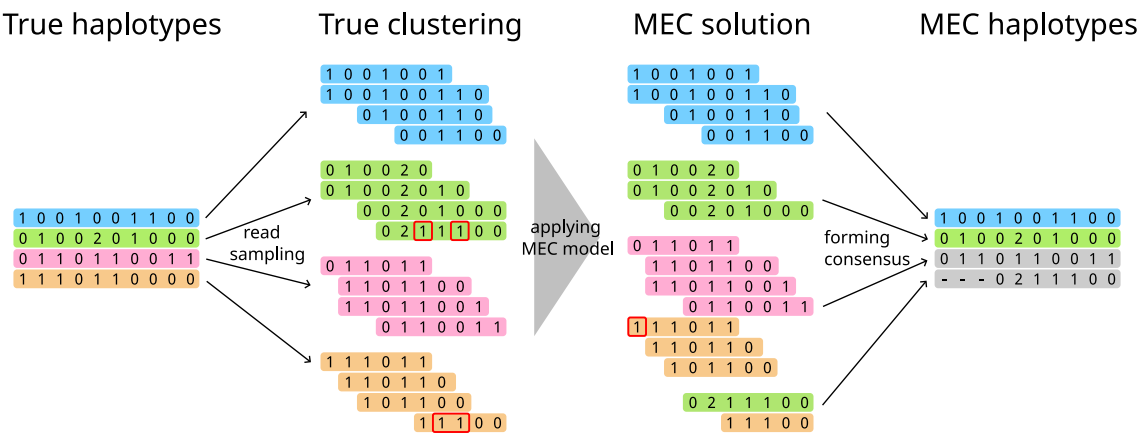


Figure 3.1: **Example of uneven MEC partitions.** Given are four haplotypes (indicated by color) with the magenta and orange ones being mostly identical. Reads are sampled with a few errors (red rectangles), resulting in a MEC score of 4 for the true read partitioning. Ignoring the expected uniform coverage distribution, the optimal MEC solution has only one error by collapsing the magenta and orange partitions into one and collecting the remaining erroneous reads into the vacant cluster. The result only contains one copy of the two identical haplotypes.

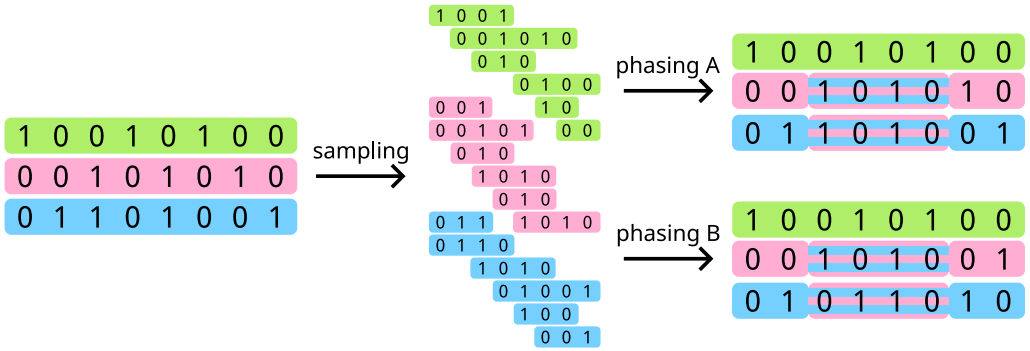


Figure 3.2: **Example of an unresolvable region.** Reads are sampled from three haplotypes (indicated by different colors) without errors. There exist two different phasings, into which all reads align perfectly and with uniform coverage. The four variants in the middle form a collapsed region of the magenta and blue haplotype and – without reads bridging the region – the phasing becomes ambiguous.

positions but no magenta or blue read is long enough the cover alleles on both left and right sides of the duplicated region. Even with error-free reads, we end up with two different but equally good options to join the magenta and blue haplotype pieces outside of the affected region. The correct strategy would be to split the phasing at the collapsed region (e.g. at the end) and report the pieces on the left and right as separate phasing blocks, even though all variants in this example are connected by reads. To our knowledge, however, most if not all existing phasing methods do not explicitly treat collapsed regions as potential cut positions.

3.1.2 Overview of existing methods

The review of existing methods up until 2019 was taken from [14]. I extended this section with more recent algorithms.

Over the last years, several polyploid phasing methods have already been proposed: In 2013, Aguiar et al. were the first to introduce a theoretical framework for polyploid haplotype assembly with the model HapCompass [77, 78], which is based on spanning trees and uses the

Minimum Weighted Edge Removal (MWER) criterion. In 2014, Berger et al. introduced HapTree [60], a maximum likelihood approach to discover the most likely haplotypes given aligned read data. To address the problem of computational complexity, HapTree assembles the most likely haplotypes for a small set of SNP positions first and then iteratively extends them while only keeping the most likely sub-solutions in each step. HapTree was shown to outperform HapCompass in terms of both accuracy and runtime [59, 60]. Together with SDhaP [79], a semi-definite programming approach based on an approximate MEC criterion, HapCompass and HapTree were evaluated and compared to each other in a simulation study conducted by Motazedi et al. [59] in 2017. The study, where simulated data of the tetraploid potato genome as model organism was used, revealed that - out of the compared methods - HapTree provided the best precision. However, it also had the highest time and memory requirements and often suffered from low recall. SDhaP showed low performance in regions of locally similar haplotypes, which is probably related to the underlying MEC model. For ploidies above 6, HapCompass was the only implementation to remain stable, although it showed an overall poor performance. As a result, none of the methods were deemed to be applicable for practical use due to computational inefficiency that prohibits scaling to large genomic regions as well as frequent failures and low overall accuracy [59].

H-PoP [40] was shown to outperform these previous approaches both in accuracy and runtime and – since then – has been considered as the state-of-the-art method. It consists of a model called Polyploid Balanced Optimal Partition (PBOP), which creates p partitions of sequence reads to minimize two measures: (i) Reads from one partition are supposed to be equal on as many variant loci as possible, whereas (ii) reads from different partitions should contain as many differences as possible. For $p = 2$, this equals the diploid MEC model, and can thus be seen as a polyploid generalization of MEC. When genotype information is present, these constraints are added to the model. The appropriate extension is then referred to as H-PoP-G.

Further advances have not proven to be useful for whole-genome single-individual haplotyping, like PolyHarsh [80], a Gibbs sampling method that is also based on the MEC model and has only been shown to work on very small artificial examples, and SDA [81]. The latter provides two algorithms based on a discrete matrix completion approach and correlation clustering, respectively, and is used to resolve segmental duplications of higher ploidy during genome assembly. However, it is not designed to scale to the whole genome.

Other matrix-based models are SCGD-hap [82], a structurally constrained gradient descent approach, and AltHap [83], which builds on SCGD-hap and aims to solve an iterative sparse tensor decomposition problem. The latter yielded results similar to those of H-PoP but also relies on MEC.

Two additional tools have been proposed that do not work well with long-read data. The work by Siragusa et al. [84] is based on minimum fragment removal which would lead to the removal of too much data considering the usually high error rates in long reads. Ranbow [85] uses allele co-occurrences on small sets of sampled positions in overlapping short reads. This

approach is susceptible to high error rates found in long reads as well because it seeds the phasing on local partitions of reads based on their allele combination on the small position samples. Thus, a large portion of the reads are clustered incorrectly and a lot of overlapping position samples are required to correct these mistakes.

A different approach to infer haplotypes is pursued by PolyCluster [86]: The idea is to convert the readset into a graph, where each node represents one read and the edge weights between two nodes are derived from the similarities of the associated read pairs. Similarly to MEC, the goal is to find a p-partition of the nodes such that a certain objective is optimized. As a metric, the authors propose the Minimum Fragment Disagreement (MFD), which maximizes the intra-cluster similarity and minimizes the inter-cluster similarity in a single function. They point out that their optimization problem is similar to correlation clustering [87] and cluster editing [88, 89] with a fixed number of clusters, both of which are NP-hard problems [87, 89, 90]. Besides an integer linear program, they also developed an approximation algorithm that first computes small local clusters to assign all nodes to a cluster, and then merges these clusters until the target number p is reached.

In 2020, we published the first version of our polyploid phasing tool, called WHATSHAP POLYPHASE [14]. Like PolyCluster, it uses local read similarity to partition a read graph, but also contains consequent steps to obtain the final haplotypes. We refer to Section 3.1.3 here, where we will elaborate further on the design choices of our method. The concept of read graphs is also adapted in FLOPP [15], in which Shaw and Yu present a different clustering objective and statistical models for similarity estimation between reads to refine local clusterings for a more even coverage distribution. This circumvents the issues of the MEC model when dealing with collapsed regions. The objective for their algorithm FLOPP is called *min-sum max tree partition*, seeking a p-partition of the nodes, such that all partitions are connected and the total weight of all partitions' maximum-weighted minimum spanning trees is minimized. In their experiments, the authors show that FLOPP outperformed H-PoP-G and the first version of WHATSHAP POLYPHASE. The latter achieved surprisingly bad results, which motivated us to continue the development of our method and improve its consistency over a wider range of input data. We will further discuss the findings of Shaw and Yu in Section 3.5.

Two other recent methods utilize both short and long reads in their computations to combine the strengths of both technologies: nPhase [91] and HAT [92]. The former aligns both read sets to a common reference and uses the short reads to identify possible variant sites. This step can be omitted if one has no short read data available but an existing variant call with genotype information instead. The long reads are then clustered in an agglomerative manner: The two most similar reads are merged at a time to form a new longer read with the consensus of the underlying reads as its new allele sequence. This process terminates when the similarity of the next merge falls below a certain threshold to avoid all reads always ending up in one large cluster. A notable property of this method is that it is ploidy-agnostic, i.e. no step requires knowledge of the target ploidy. The second tool performs a seeding step on the short reads

resulting in very short but accurate phasing blocks. The long reads are consequently threaded through these blocks to identify the best linkage between them.

Saada et al. [93] assembled an overview and classification of polyploid phasing methods. They defined four categories of reference-based algorithms: (i) population inference, (ii) objective function optimization, (iii) graph partitioning, and (iv) cluster building. The second category covers the MEC-based methods described above, while the third category is represented by PolyCluster, FLOPP, and WHATSHAP POLYPHASE. HAT and nPhase fall into the last category, as they compute partitions without utilizing global graph models.

To our knowledge, none of the previously existing methods give reliable information about the accuracy of the resulting haplotypes since they are either output in one consecutive sequence or in very long blocks. In particular, this means that there is no information about the likely positions of switch errors. Thus, large regions of the resulting haplotypes might be incorrect, but it is not possible to identify these regions, which makes the results very difficult to use in practice.

3.1.3 Outline of WHATSHAP POLYPHASE

The observation that collapsed regions are an exclusive phenomenon in polyploid genomes motivated us to design a model that explicitly allows shared sequences between haplotypes. We decided to depart from the commonly used MEC model and moved towards a graph clustering model, similar to PolyCluster [86]. The full algorithm itself evolved over time and the most recent version deviates from the original algorithm described in [14]. This overview will focus on the latest state of the algorithm as it is implemented in the WhatsHap software package. While presenting the individual steps, we will point out major differences to the original algorithm as well as the motivations behind them.

WHATSHAP POLYPHASE is organized as a pipeline that consists of three consecutive stages: the clustering stage, the threading stage, and the reordering stage. As described in Section 1.4, the input for the phasing problem is an allele matrix $\mathcal{A}$, a ploidy p, and genotypes $G_1, \dots G_m$. The output is primarily the p sequences that encode the taken alleles for each haplotype, but as pointed out in Section 1.4.1, the secondary output is a set of cut positions that divide the phasing into blocks. Ideally, there would be only one block for the entire input region but, in practice, the input reads usually do not allow for a contiguous phasing over long distances. Especially the before-mentioned collapsed regions are a great source of uncertainty in the polyploid case.

Figure 3.3 illustrates the different steps of our method: From the input allele matrix, we create a graph with one node for each read and a weighted edge for each overlap between two reads. The weight depends on the similarity of the two associated reads to encode how likely these reads should be assigned to the same haplotype. Using the cluster editing model [88], we cluster all reads according to their supposed haplotype origin to form “puzzle pieces” that are assembled in the threading stage (Section 3.2). Given an input ploidy p, we obtain

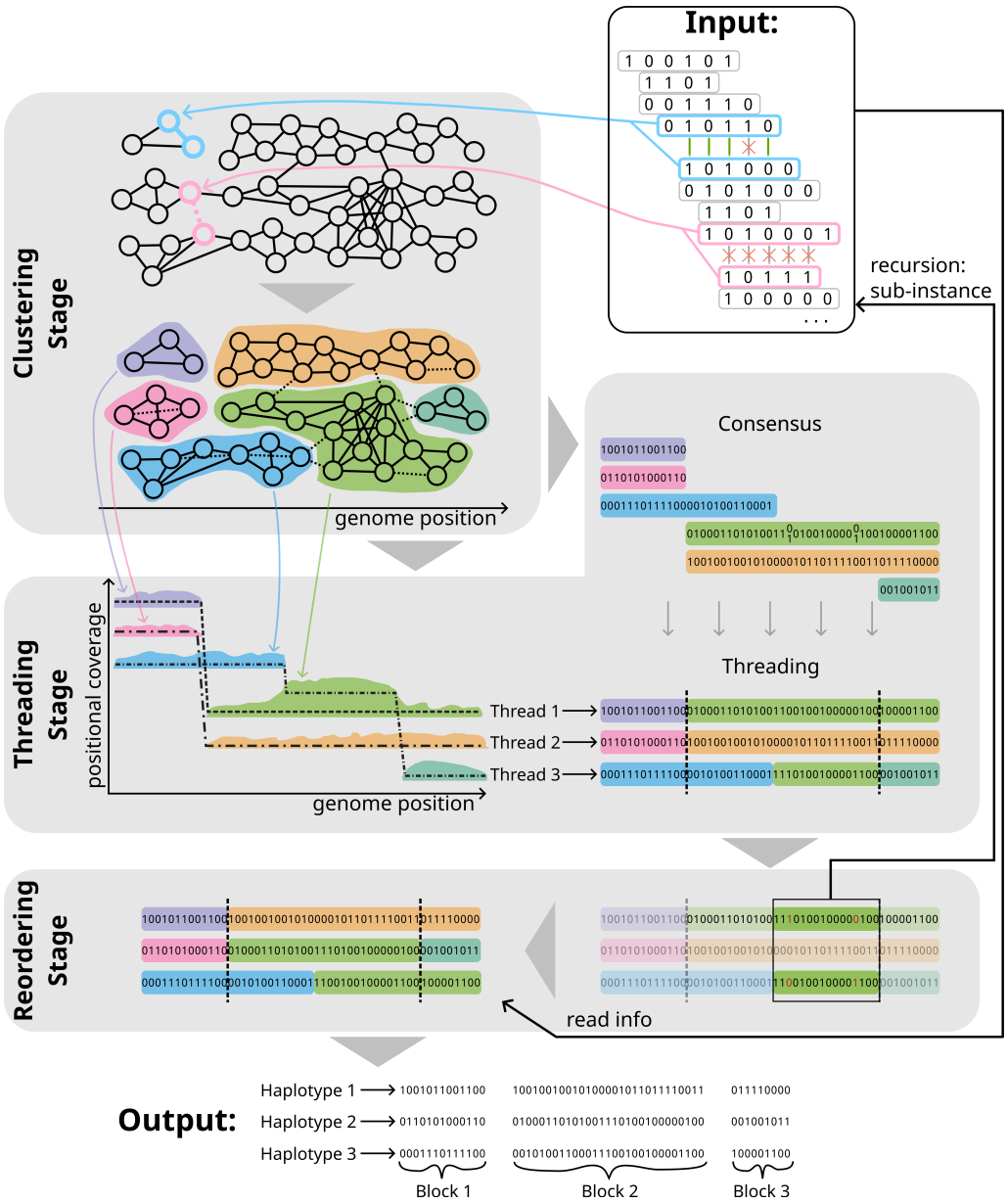


Figure 3.3: **Overview of WhatsHap Polyphase.** Reads are represented as rows of an input allele matrix and pass three stages. Stage I: Statistical scoring of each read pair classifies them into belonging to the same (blue pair) or different haplotypes (magenta pair). Associated reads are indicated by drawn edges, while dissociation and absence of overlap are both indicated by missing edges for better visual distinction. Stage II: Clusters (colored shapes) are aligned to their spanned genomic regions and receive a position-wise height that is proportional to their relative coverage among all clusters. Computed threads (three black lines with different dash patterns) represent mosaics of clusters to explain cluster location and coverage. Cluster switches give hints to possible cut positions (two vertical dashed lines). Stage III: Resolves alleles inside collapsed regions by creating sub-instances that are solved recursively. Read information from the input facilitates the reordering of haplotype blocks between the cut positions.

p paths (called *threads*) through the computed read clusters, which induce p allele sequences when adding the position-wise consensus of each cluster (Section 3.3). In the reordering stage, we resolve genomic variation inside collapsed regions (if present) by applying our algorithm recursively on just the affected region and corresponding reads. After several refinement steps, the p allele sequences are cut alongside any uncertain position and reported as final output

(Section 3.4). Throughout the following sections, we will describe each of the three stages in detail.

3.2 Read clustering

The clustering idea is based on [14]. The read scoring method is new and differs from the previously published one. The methods are presented in more detail here compared to [14].

The first step in our phasing pipeline is to compute a clustering of all reads intending to identify groups of reads that likely originate from the same haplotype. Each of the p original haplotypes carries some characteristic genetic variation that is passed on to the sampled reads, allowing us to estimate the likelihood of two reads being sampled from the same haplotype or not. Ideally, the read set is separable into p partitions that correspond to the original haplotypes.

This rationale of computing a p-partition based on pairwise comparisons between reads has been quite common among other polyploid phasers in the past few years [15, 86, 91]. However, the presence of collapsed regions impedes the success of finding a global p-partition, as we have already seen in the example from Figure 3.2. We believe that a dynamic number of clusters is more suited to capture the nature of polyploid genomes for two reasons: First, reads from multiple haplotypes can be clustered together inside a collapsed region; this is a more accurate representation than forcing those reads to be split because of a fixed target cluster count. Second, if two clusters of reads are dissimilar from each other, we should retain this information and not forcibly join the clusters to achieve a fixed cluster count. Instead, we perform a separate step, namely the *threading stage*, that computes haplotype sequences based on read clusters. This separation allows us to identify unresolvable sites in collapsed regions and insert appropriate cut positions.

3.2.1 Cluster editing

We chose *cluster editing* as the underlying clustering model [88, 89, 94]. It takes an undirected graph $G = (V, E)$ as input and computes a minimum-cost set of edge modifications to transform G into a *clique graph*. That is all connected components in the transformed graph are fully connected subgraphs and represent the clusters.

More formally, let V be a set of nodes and $\binom{V}{2}$ be the set of node pairs over V. We denote node pairs $\{u, v\}$ as uv with $uv = vu$. Let further $c : \binom{V}{2} \to \mathbb{R}$ be a function that assigns a weight to each node pair and let $G = (V, E)$ be an undirected graph with $E = \{uv \in \binom{V}{2} : c(uv) > 0\}$ containing all node pairs with a positive weight, the so-called *existing* edges. The cost of removing an existing edge from G equals its weight. On the contrary, all node pairs $\binom{V}{2} \setminus E$ form the *missing edges*, where each missing edge uv can be inserted into G for cost $-c(uv) = |c(uv)|$. Note that we treat edges with weight 0 – so-called *zero-edges* – as missing edges in G that can be inserted for free if needed. We will use the edges and node pairs as interchangeable terms.

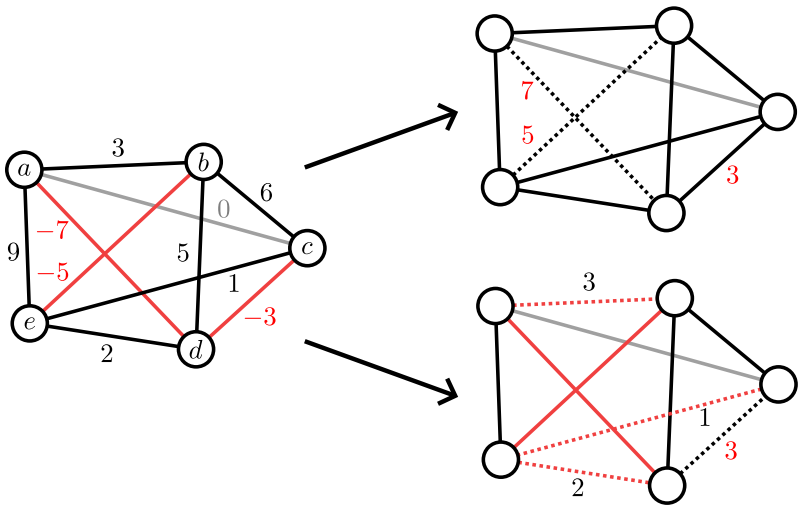


Figure 3.4: **Example for cluster editing.** Left: Input graph for cluster editing. Present edges with deletion cost are drawn in black and missing edges with insertion cost are drawn in red. The gray edge is a zero-edge. Right: Two possible solutions for cluster editing. Dashed lines indicate edge modifications with the numbers showing the actual paid cost for each change. The bottom graph is the optimal transformation with a cost of 9, while the top graph is suboptimal with a cost of 15.

A node triple $u, v, w \in V$ is called a *conflict triple* if edge transitivity between these nodes does not hold, i.e. $uv, vw \in E$ but $uw \notin E$. G is called a clique graph, if it does not contain any conflict triples. This definition is equivalent to the more intuitive description of G consisting of only disjoint cliques: For any node triple u, v, w, the nodes inside the same clique are always fully connected, while nodes from different cliques are never connected, effectively forbidding the conflicting structure $uv, vw \in E$ but $uw \notin E$. Cluster editing can finally be described as the following combinatorial optimization problem:

Problem 5 (Cluster editing). *Given an undirected graph $G = (V, E)$ and a weight function $c : \binom{V}{2} \to \mathbb{R}$, find a minimum-cost transformation of G into a clique graph $G' = (V, E')$ by deleting edges from E or inserting edges not in E. The transformation cost $c(G, G')$ is defined as the sum of the absolute weights of all modified edges, i.e.*

$$c(G, G') = \underbrace{\sum_{e \in E \setminus E'} c(e)}_{\text{deletion cost}} - \underbrace{\sum_{e \in E' \setminus E} c(e)}_{\text{insertion cost}}. \tag{3.1}$$

An example of a cluster editing instance is given in Figure 3.4. It shows a graph with five nodes and ten potential edges: six existing edges, three missing edges, and one zero-edge. The latter can act as an existing or missing edge without causing modification costs. The conflict triples abd, abe, ade, bcd, bce, bde and cde have to be resolved. The example gives two possible solutions. The first inserts all missing edges to form a single clique for modification cost 15, while the second solution splits the graph into two cliques for a total cost of 9.

In order to apply the clustering model to the given allele matrix (and thus clustering the reads therein), we first have to transform the matrix into a suitable graph with a weight function over all pairs of nodes. For the transformation, edge weights have to correlate with haplo-

type co-occurrence of their respective endpoints, i.e., a positive weight for reads rather belonging to the same haplotypes and a negative weight for the opposite case. For two reads with sufficiently many common variants, their allele similarity gives evidence for either of these two cases. As a result, we expect the graph to form almost-cliques of positively connected nodes (i.e. nodes connected by actual edges or edges with positive weight) for those reads belonging to the same haplotype, while these almost-cliques themselves are mostly disconnected from each other (or have negative edge weights between them). Since we cannot expect clusters to be perfectly separated due to sequencing errors, mapping errors, or local similarities between different haplotypes, we apply the weighted cluster editing model to find the cheapest way of resolving all conflicts in terms of modification cost.

3.2.2 Pairwise scoring

The goal of defining a proper cost function for cluster editing is not only to decide whether two reads should be clustered together but also to determine the strength of the evidence. In [14], the two key elements of the model are the allele error rate, which is the probability of wrongly calling a single allele for a single read, and the dissimilarities between the original haplotypes.

First, we compute a sorted list of all dissimilarities between read pairs, i.e. the fraction of different alleles between two reads in relation to the total number of common variants (not counting the gap positions for any of the two reads). Second, we use two observations: (i) All haplotypes are equally abundant among the reads, which means that $\frac{1}{p}$ of all read pairs are expected to contain two reads of the same haplotype origin, and (ii) the dissimilarity of read pairs belonging to the same haplotype is expected to be significantly lower than for those belonging to different haplotypes because the latter are affected by differences between haplotypes.

We then assume that the lowest $\frac{1}{p}$ dissimilarities from the sorted list correspond to read pairs with both reads stemming from the same haplotype. We use the average dissimilarity d_{same} over this group as an estimator of the allele error rate. In reverse, the upper $\frac{p-1}{p}$-fraction of dissimilarities represents the opposite case of different haplotype origins within the read pairs. These dissimilarities are a combination of the allele error rate and the average dissimilarity between different haplotypes. We call the average over these dissimilarities d_{diff}.

To determine the edge weight between two reads R_i and R_j we first count the number of shared variables and the number of different alleles – which we already did to compute all the dissimilarity values. After that, we compute the likelihoods of observing these counts when assuming d_{same} and d_{diff} as the probability of position-wise allele disagreement. Using two binomial distributions with the respective probabilities as parameters, we obtain one likelihood for the read pair belonging to the same haplotype and one likelihood for the opposite case. Finally, the weight is defined as the logarithm over the first likelihood divided by the second one. A positive weight then means that the common haplotype is more likely, while a negative weight is the result of different haplotypes being more likely.

A weakness of this model is the assumption that all $\binom{V}{2}$ pairs of haplotypes exhibit the same degree of dissimilarity, which is rather unlikely. In fact, it would have been more accurate to estimate dissimilarities for all pairs of haplotypes and interpret the list of read pair dissimilarities as an aggregation over $\binom{V}{2}$ distributions.

In an attempt to improve the phasing quality of the method in [14], we present a more rigorous approach following Bayesian statistics: Let $R_i, R_j \in \mathcal{R}$ and let $h : \mathcal{R} \to \{1, \dots, p\}$ be a function that maps a read to its true haplotype. We define the score $s(R_i, R_j)$ of R_i and R_j as

$$s(R_i, R_j) := \log\left(\frac{P\left(h(R_i) = h(R_j) \mid R_i, R_j\right)}{P\left(h(R_i) \neq h(R_j) \mid R_i, R_j\right)}\right). \tag{3.2}$$

For each hypothesis (that R_i and R_j are from the same or different haplotypes) we want to compute the likelihood given the read sequences themselves. If R_i and R_j are more likely to share a haplotype, the likelihood ratio will be greater than 1 and, through the logarithm, result in a positive score. Analogously, if R_i and R_j rather belong to different haplotypes, the ratio will be less than 1 and lead to a negative score. A score of zero would be interpreted as both hypotheses having the same likelihood, and thus total insecurity about the reads' relationship. This is in line with the cluster editing model, where positive weights describe costs to remove an existing edge and negative values describe costs to insert a missing edge. A weight of zero therefore allows us to interpret an edge both ways without any costs.

Next, we use the Bayesian theorem to compute the likelihood of the reads given their haplotype origin:

$$\frac{P\left(h(R_i) = h(R_j) \mid R_i, R_j\right)}{P\left(h(R_i) \neq h(R_j) \mid R_i, R_j\right)} = \frac{\frac{P(R_i, R_j \mid h(R_i) = h(R_j)) \cdot P(h(R_i) = h(R_j))}{P(R_i, R_j)}}{\frac{P(R_i, R_j \mid h(R_i) \neq h(R_j)) \cdot P(h(R_i) \neq h(R_j))}{P(R_i, R_j)}} \tag{3.3}$$

The prior probabilities $P(h(R_i) = h(R_j))$ and $P(h(R_i) \neq h(R_j))$ do not depend on the reads' alleles. For a given read, we assume that every haplotype is equally likely to be its origin. Thus, we derive $\frac{1}{p}$ and $1 - \frac{1}{p}$ as prior probabilities and gain

$$\frac{P\left(h(R_i) = h(R_j) \mid R_i, R_j\right)}{P\left(h(R_i) \neq h(R_j) \mid R_i, R_j\right)} = \frac{P(R_i, R_j \mid h(R_i) = h(R_j))}{P(R_i, R_j \mid h(R_i) \neq h(R_j))} \cdot \frac{p}{1 - \frac{1}{p}}. \tag{3.4}$$

It remains to be shown how to compute the reverse likelihoods $P(R_i, R_j \mid h(R_i) = h(R_j))$ and $P(R_i, R_j \mid h(R_i) \neq h(R_j))$. If given the true haplotypes and an estimate on the allele error rate (i.e. the probability for a single allele on a single read to be detected wrongly such that it looks like another allele), we could enumerate all combinations of haplotype origins for both reads, compute the likelihood to sample each of the reads from the selected haplotypes (respecting the allele error rate), and aggregate them in a sum.

Since the original haplotypes are not available, however, we would rather have to enumerate all *possible* haplotype configurations (constrained by the given genotypes) and repeat the procedure for all these configurations. This would be computationally infeasible because

a region of length k would have $\Omega(p^k)$ many possible phasings, given that any heterozygous genotype allows for at least p different phasings for one variant and we got k independent variants in our region.

As relaxation of our proposed model, we assume all variants of the overlap region $Ov(R_i, R_j)$ of R_i and R_j to be independent, even though this is generally not the case. This allows us to decompose the reverse likelihood into a product over all variants in $Ov(R_i, R_j)$ as shown in Equations 3.5 to 3.10.

$$P(R_i, R_j \mid h(R_i) = h(R_j)) \tag{3.5}$$

$$\simeq \prod_{l \in Ov(R_i,R_j)} P(R_i[l], R_j[l] \mid h(R_i) = h(R_j)) \tag{3.6}$$

$$= \prod_{l \in Ov(R_i,R_j)} \left(\sum_{a \in G_l} \frac{f_a(G_l)}{p} \cdot P(R_i[l], R_j[l] \mid H_{h(R_i)}[l] = H_{h(R_j)}[l] = a) \right) \tag{3.7}$$

$$P(R_i, R_j \mid h(R_i) \neq h(R_j)) \tag{3.8}$$

$$\simeq \prod_{l \in Ov(R_i,R_j)} P(R_i[l], R_j[l] \mid h(R_i) \neq h(R_j)) \tag{3.9}$$

$$= \prod_{l \in Ov(R_i,R_j)} \left(\sum_{a,b \in G_l, a \neq b} \frac{f_a(G_l) \cdot f_b(G_l)}{p(p-1)} \cdot P(R_i[l], R_j[l] \mid H_{h(R_i)}[l] = a, H_{h(R_j)}[l] = b) \right) \tag{3.10}$$

Finally, we derive the input graph for cluster editing by taking the read set $\mathcal{R}$ as the node set V and adding edges for any positively-scored read pair, i.e. $E := \{R_i R_j \mid s(R_i, R_j) > 0\}$. As a cost function, we use the values of the scoring scheme s, i.e., $c(R_i, R_j) := s(R_i, R_j)$. All non-overlapping pairs of reads are assigned a score of 0. We interpret these pairs as missing edges in the graph that can be inserted for no cost. Note that zero-edges are still added to the modified edge set E' upon insertion, even if they do not influence the total transformation cost.

3.2.3 Clustering algorithm

Cluster editing has been shown to be NP-hard in general, even for the unweighted version [88]. In previous publications, there have been several proposals on how to solve this model [94, 95]. These include specifically engineered algorithms as well as Integer Linear Programming formulations. Additionally, a variety of reduction rules has been developed to shrink down instances without altering their effective solution. Experiments on generated and real instances have shown that these exact strategies were able to cluster graphs with up to a few thousand nodes [95, 96]. For clustering an entire chromosome, however, the instance sizes are many orders of magnitudes higher, ranging from tens of thousands of reads up to millions of reads. It became clear quite quickly that solving such huge instances to optimality is not feasible. In fact, even our heuristic approach required a lot of tuning to reach practically acceptable

runtimes. We therefore abandoned further research on faster exact methods and focused on scalability to larger instances instead.

The backbone of the final heuristic is a greedy algorithm that selects one edge from the graph at a time and makes a decision whether the two endpoints should end up on the same or different clusters. Depending on this decision, the edge is then either taken for or excluded from the final graph G'. For consistency with the edge selection process, we set the weight of a taken edge to ∞ and call this edge *permanent*, as it can never be removed from the solution again. In reverse, we set the weight of an excluded edge to $-\infty$ and call this edge *forbidden*. The greedy algorithm proceeds, until all edges in the graph are either permanent or forbidden.

Induced costs for edge selections

As criteria for selecting edges and choosing their role, we introduce *induced costs* for each edge uv, stating the immediate cost for the clustering model if uv is made permanent or forbidden. These costs are called $\text{icp}(uv)$ and $\text{icf}(uv)$ respectively and were originally defined in [95].

For a node triple u, v and w (short: uvw), the induced cost of adding uv to the solution depends on whether $c(uv)$ is negative and whether either exactly one out of $c(uw)$ and $c(vw)$ is negative. If $c(uv)$ is negative, we obviously have to pay for inserting uv. If $c(uw)$ and $c(vw)$ are both negative or both positive, there is no conflict triple after inserting uv. Otherwise, we have to either insert the missing edge or remove the present edge (out of uw and vw), whatever is cheaper. Mathematically, this can be written as:

$$\text{icp}_w(uv) = \max(0, -c(uv)) + \begin{cases} \min(|c(uw)|, |c(vw)|) & \text{if } c(uw) \cdot c(vw) < 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.11}$$

The induced costs of excluding uv from the solution functions similarly, but costs arise from $c(uv)$ being positive and from $c(uw)$ and $c(vw)$ being both positive to form a conflict triple:

$$\text{icf}_w(uv) = \max(0, c(uv)) + \begin{cases} \min(c(uw), c(vw)) & \text{if } c(uw), c(vw) > 0, \\ 0 & \text{otherwise.} \end{cases} \tag{3.12}$$

The total induced costs for an edge uv is the sum over all triple-based induced costs:

$$\text{icp}(uv) = \sum_{w \in V \setminus \{u,v\}} \text{icp}_w(uv) \tag{3.13}$$

$$\text{icf}(uv) = \sum_{w \in V \setminus \{u,v\}} \text{icf}_w(uv) \tag{3.14}$$

The runtime for computing all induced costs is $\mathcal{O}(n^3)$ with n being the number of nodes because there are $\frac{n \cdot (n-1)}{2}$ many edges in the graph and we need to consider $(n-2)$ triangles for each edge. Since zero-edges can never be part of a conflict triple, the computation time for

induced costs can be drastically reduced by only considering neighbors w with $c(uw) \neq 0 \neq c(vw)$, because their contribution to the total induced cost will always be 0. Zero-edges are used for non-overlapping read pairs, which, considering that reads are very short compared to a chromosome, comprise the vast majority of read pairs. Let k be the number of non-zero-edges in G. Each non-zero-edge uv is part of $(n-2)$ triangles in the graph, so it is considered at most $(n-2)$ times for the icp_w and icf_w computation (in practice even much less because many of these triangles will contain another zero-edge and thus be omitted). This reduces the worst-case runtime to $\mathcal{O}(n \cdot k)$.

When updating an edge's weight – say of edge uv – we do not need to re-compute all induced costs for the graph because every edge is part of at most $n-2$ triangles, and can thus only influence the induced cost of at most $2 \cdot (n-2)$ other edges (namely edges uw, vw for each $w \in V \setminus \{u, v\}$). Moreover, for each influenced edge uw (or vw) we do not need to repeat the entire computation for $\text{icp}(uw)$ and $\text{icf}(uw)$, but only for $\text{icp}_v(uw)$ and $\text{icf}_v(uw)$. We subtract the old contributions of $\text{icp}_v(uw)$ and $\text{icf}_v(uw)$ from $\text{icp}(uw)$ and $\text{icf}(uw)$, respectively, and add the new contribution based on the updated weight of uv. In total, the update of induced costs can be done in $\mathcal{O}(n)$ time. In practice, it is usually much less, as we do not need to update triangles with a zero-edge after updating uv's weight.

Algorithmic outline

For simplicity, we assume that the original graph only contains real-valued weights, i.e., no "∞" or "$-\infty$". Thus, all induced costs are real-valued as well. As a first step, we identify the highest induced cost and the associated edge uv. If $\text{icp}(uv) > \text{icf}(uv)$, this means that including uv into the solution would be the most expensive operation, accounting for the immediate cost only. Therefore, we choose to exclude uv, that is, we make uv forbidden by setting $c(uv)$ to $-\infty$. Analogously, if $\text{icf}(uv) > \text{icf}(uv)$, we make uv permanent and set $c(uv)$ to $+\infty$. If both costs are identical, we exclude uv. This process is repeated until all edges are either permanent or forbidden.

Our implementation utilizes two heaps to store all icf and icp values, so we can always extract its maximum in constant time. Once an edge becomes permanent or forbidden, we remove it from both heaps to not process the same edge a second time. For the initial computation, we need $\mathcal{O}(n^2 \cdot \log n)$ time for sorting and building the heap. The update cost for a single edge modification is increased to $\mathcal{O}(n \cdot \log n)$ to account for sifting operations inside the heap.

Algorithm 1 summarizes the described procedure. The runtime is dominated by the while-loop, as it loops over all $\mathcal{O}(n^2)$ potential edges with a worst-case running time of $\mathcal{O}(n \cdot \log n)$ for each iteration and $\mathcal{O}(n^3 \cdot \log n)$ in total. Regarding correctness, we still have to prove that our heuristic computes a valid clique graph, i.e. all conflict triples in G are eliminated. We will prove the following lemma by induction and then conclude the validity of the heuristic output:

Algorithm 1 Cluster editing heuristic

Input: Graph $G = (V, E)$ with cost function c for all edges
Output: G clustered into cliques
  compute initial icf and icp values
  create heaps over icf and icp values
  while heap_{icf} is not empty **do**
    $uv \leftarrow \arg\max(\text{heap}_{\text{icp}})$
    $u'v' \leftarrow \arg\max(\text{heap}_{\text{icf}})$
    if $\text{icp}(uv) \geq \text{icf}(u'v')$ **then**
      $c(uv) \leftarrow -\infty$
      Remove uv from both heaps
    else
      $c(u'v') \leftarrow \infty$
      Remove $u'v'$ from both heaps
    end if
    update induced costs
  end while
  report connected components as clusters

Lemma 1. *At the beginning of each while-loop iteration of Algorithm 1, there exists no node triple u, v, w, such that uv and vw are permanent and uw is forbidden. That means that there is no conflict triple induced by already-processed edges.*

For a direct proof of this lemma, we need some auxiliary statements first. Let $C_1, \ldots, C_q \subseteq V$ be the connected components of G induced by only the permanent edges, and let $C(u)$ be the connected component that contains node u. For the first iteration, each node is its own component and larger components are formed once the algorithm inserts permanent edges. We will first show that the algorithm prioritizes the completion of connected components into cliques of permanent edges.

Lemma 2. *Let $u, v \in V$ with $C(u) \neq C(v)$ and uv being unprocessed. If the algorithm makes uv permanent in the next iteration (i.e. $C(u)$ and $C(v)$ are merged) and all edges within $C(u)$ and $C(v)$ already are permanent, the algorithm will consequently make all node pairs wx with $w \in C(u)$ and $x \in C(v)$ permanent as well. The merged component $C(u) \cup C(v)$ will thus form a clique of permanent edges.*

Proof. Since the algorithm decided to make uv permanent, we know that $\text{icf}(uv) > \text{icp}(uv)$. From $C(u) \neq C(v)$, we conclude that $\text{icf}(uv) < \infty$ because this could only happen if u and v had a common neighbor w with $c(uw) = c(vw) = \infty$, contradicting the prerequisite.

Let $E_u = \{uw \in \{u\} \times (C(v) \setminus \{v\})\}$. No node pair $uw \in E_u$ can be forbidden because with vw being permanent the induced cost $\text{icp}(uv)$ would be infinite in such a case – a contradiction to the algorithm's choice. After uv becomes permanent, the induced cost $\text{icf}(uw)$ becomes infinite, resulting in the algorithm making all node pairs in E_u permanent as an immediate reaction. The same holds for the symmetric set $E_v = \{wv \in (C(u) \setminus \{u\}) \times \{v\}\}$. Thus, all potential edges incident to uv become permanent.

Let $E_r = \{wx \in (C(u) \setminus \{u\}) \times (C(v) \setminus \{v\})\}$ be the remaining node pairs between $C(u)$ and $C(v)$. For every $wx \in E_r$ the node pairs uw and ux are already permanent. Thus, the induced cost icf(wx) would be infinite and the algorithm will consequently set all node pairs from E_{uv} to permanent as well in the following iterations. As a result, all edges between $C(u)$ and $C(v)$ become permanent after uv such that $C(u) \cap C(v)$ forms a clique of permanent edges.

□

Similarly, the algorithm consequently forbids all edges between two connected components C_i, C_j if one edge between C_i and C_j is set to forbidden.

Lemma 3. *Let $u, v \in V$ with $C(u) \neq C(v)$ and uv being unprocessed. If the algorithm makes uv forbidden in the next iteration, it will forbid all node pairs wx with $w \in C(u), x \in C(v)$.*

Proof. Using Lemma 2, we know that $C(u)$ and $C(v)$ form cliques of permanent edges because whenever two components are permanently connected, the algorithm immediately adds all missing permanent edges to restore transitivity. We use a similar technique as for Lemma 2: Let $E_u = \{uw \in \{u\} \times (C(v) \setminus \{v\})\}$. After uv becomes forbidden, the induced costs icp(uw) for every $uw \in E_u$ becomes infinite because vw already is permanent. Therefore, the algorithm will immediately forbid all node pairs from E_u. Following the same pattern as the proof for Lemma 2, all node pairs from $E_v = \{wv \in (C(u) \setminus \{u\}) \times \{v\}\}$ and $E_r = \{wx \in (C(u) \setminus \{u\}) \times (C(v) \setminus \{v\})\}$ become forbidden as well in the following iterations, concluding the proof.

□

Using Lemma 2 and 3, we can finally prove Lemma 1: Starting without permanent or forbidden edges with all nodes in their own connected component, the algorithm selects one edge, leading to either the situation in Lemma 2 (permanent edge) or the one in Lemma 3 (forbidden edge). Separating or connecting two components does not introduce new conflict triples. With the algorithm immediately restoring transitivity concerning permanent or forbidden edges, there can never be a situation where a conflict triple among permanent or forbidden edges is formed. In summary, Algorithm 1 computes a consistent clustering, although without quality guarantees. We denote the clustering as $\mathcal{C}$ and the clusters as $C_1, \ldots, C_q$.

3.2.4 Cluster refinement

The presented cluster refinement was part of the old WHATSHAP POLYPHASE *algorithm presented in [14], but this step was not explained there. I added a description and illustration of how the refinement works.*

The new version of WHATSHAP POLYPHASE performs a single pass of Algorithm 1 to obtain a read clustering. The old version, however, uses multiple passes to refine the results from this stage. We will briefly sketch this refinement step because it will be relevant for the performance analysis of the old and new versions.

One reason to allow an arbitrary cluster count was to collect reads of multiple haplotypes in one cluster within a collapsed region. However, the haplotype sequences are not necessarily

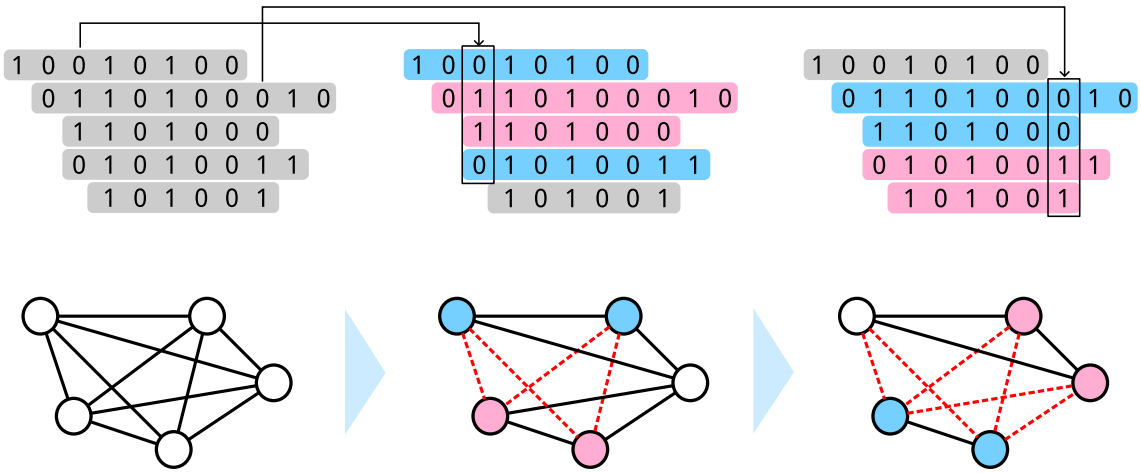


Figure 3.5: **Cluster refinement.** The five sequences correspond to a cluster of five reads with two variants showing significant allele disagreement. For both variants, the spanning reads can be split into two partitions (blue and magenta). The graph below the sequences represents the read subgraph for just these five reads. Initially, all reads were connected by positive weights (straight black lines), but alongside the detected allele splits, subsets of nodes are separated from each other by defining the weight as forbidden (dashed red lines).

identical within such a region but might contain very few differences, which allows us to split a cluster by haplotype origin after the cluster editing model was solved. Figure 3.5 shows an example of a read cluster containing reads from two different haplotypes. This is indicated by two variants with an allele split of about 1:1 within the cluster. For each of both variants, we can split a subset of the reads by their supported allele, resulting in two partitions (blue and magenta) each.

The idea of the cluster refinement is to scan each cluster for variants with a significant allele disagreement, i.e. a p-value of less than 0.02 for the hypothesis that the disagreement was caused by random allele errors, assuming a fixed allele error rate of 0.05. For every selected variant, we split the spanning reads into two partitions and modify the original read graph G such that the two partitions are fully disconnected by forbidden edges (weight set to $-\infty$). Finally, we rerun Algorithm 1 with the updated node pair weights, until no further variants with significant allele disagreement are found or until a certain number of iterations (default: 5) has been executed.

In every pass, the algorithm is forced to split clusters previously containing reads from multiple haplotypes if differences between those haplotypes exist on some pivot variants. Reads containing allele errors on these pivot variants might be wrongly forced out of their correct cluster. When designing this procedure, we expected that the majority of reads would remain error-free on the pivot variants and result in a clean split with only a few dropouts. As a consequence, some collapsed regions would already be resolved on a cluster level before the clustering was passed on to the threading stage. This refinement step was removed from the new version of WHATSHAP POLYPHASE because we introduced a separate step to resolve collapsed regions. As a side effect, we avoided repetitions of the already time-consuming clustering step and could noticeably reduce the runtime of our phasing algorithm.

3.3 Haplotype threading

The original idea of the threading model was proposed by Rebecca Serra Mari [14]. I later removed the genotype constraints, re-implemented it in C++, and added symmetry elimination to improve the runtime.

For the second part of the algorithm, we developed a novel approach called *haplotype threading* to find p allele sequences that optimally represent the read clusters from the first stage. This step is necessary because the reads are not necessarily split into exactly p clusters and the clustering model does also not promote a uniform position-wise coverage among the clusters. A single haplotype can rather be fragmented over many clusters and single clusters may hold reads from multiple haplotypes inside collapsed regions. The idea behind haplotype threading is that every haplotype is expressed as a sequence of read clusters, i.e., for every variant position, a haplotype is assigned to a single read cluster. If the clusters are plotted as "clouds" in the two-dimensional space with the x-axis as genome position and the y-axis as the position-wise coverage like in Figure 3.3, the haplotypes look like threads traversing the entire phased region, hence the name haplotype threading. We will explain this idea more formally throughout the following subsections.

3.3.1 Characterizing haplotypes as threads

Following the illustration of read clusters as clouds, we say that a cluster $C \in \mathcal{C}$ *spans* an interval of variants, defined by the first and last non-gap position of any read in C, which we call $s(C) := \min_{R \in C} s(R)$ and $e(C) := \max_{R \in C} e(R)$. For each variant v_i this gives us the set $\text{span}(v_i)$ of clusters spanning v_i. Let furthermore $\text{cov}(C, i)$ be the *absolute coverage* of C for variant v_i. That is the number of reads r in C for which $r[i] \neq -$. The *relative coverage* $\text{cov}_\text{r}(C, i) := \frac{\text{cov}(C,i)}{\sum_{C' \in \text{span}(v_i)} \text{cov}(C',i)}$ for cluster C and variant v_i indicates the fraction of total coverage contributed by C for v_i. The before-mentioned clouds in Figure 3.3 show the span of each cluster on the x-axis and its relative coverage for every position on the y-axis.

We intend to represent haplotypes as threads through the clustering. Therefore, a *thread* T is a sequence of length m over the cluster indices $1, \ldots, q$, where q is the number of clusters in $\mathcal{C}$ and we assume the contained clusters $C_1, \ldots C_q$ to be indexed in some fixed manner. With $T[i]$ we denote the selected cluster index for the i-th variant position. The goal is to compute a *threading* $\mathcal{T} = (T_1, \ldots, T_p)$ containing as many threads as our target ploidy. For a single variant v_i, the cluster indices $(T_1[i], \ldots, T_p[i])$ form a p-tuple, which we call *cluster tuple*. Thus, a threading can also be characterized as a sequence of m cluster tuples. We denote the i-th cluster tuple of $\mathcal{T}$ as $T_{(i)}$. Note that a cluster index j may occur multiple times in a cluster tuple. We call this number of occurrences the *multiplicity* of a cluster index inside a cluster tuple and write it as $\text{mult}(T_{(i)}, j)$.

3.3.2 Threading model

The computed threading must represent the clusters (and thus the reads) appropriately. We formulate two objectives the optimal threading has to maximize: (i) representative read coverage and (ii) haplotype contiguity. The first criterion ensures that the relative coverage for each cluster is reflected in its multiplicity among the cluster tuples. The expected multiplicity of a cluster C_j at v_i is given by $\text{mult}_{\text{exp}}(i,j) = \left\lceil p \cdot \text{cov}_{\text{r}}((,C),i) - \frac{1}{2p} \right\rfloor$, while the selected multiplicity $\text{mult}(T_{(i)},j)$ of C_j is given by the number of appearances of C_j in $T_{(i)}$. Deviations between expected and selected cluster multiplicities are penalized by a constant p_{cov} per cluster so that a cluster tuple $T_{(i)}$ is evaluated by the cost function

$$\text{cost}_{\text{cov}}\left(T_{(i)},i\right) = \sum_{j=1}^{p} p_{\text{cov}}[\![\text{mult}_{\text{exp}}(i,j) \neq \text{mult}(T_{(i)},j)]\!].$$

The second criterion encourages threads to stay inside the same cluster for as long as possible. Since clusters represent coherent fragments of haplotypes, we not only want the local coverage of each cluster to be considered but each cluster should be covered by as few distinct threads as possible. For two consecutive cluster tuples $T_{(i)}$ and $T_{(i+1)}$, we define the penalty of a cluster switch by p_{switch}, which results in the cost function

$$\text{cost}_{\text{switch}}\left(T_{(i)},T_{(i+1)}\right) = \sum_{j=1}^{p} p_{\text{switch}}[\![T_j[i] \neq T_j[i+1]]\!] + p_{\text{affine}} \cdot [\![T_{(i)} \neq T_{(i+1)}]\!].$$

In other words, we penalize the switch from one cluster to another on two consecutive variants inside each thread by some constant p_{switch} and every variant that contains switches in general by some affine switch cost p_{affine}. The affine cost encourages the model to co-locate switches on multiple threads if they occur in close proximity. Such events will later be used to identify candidates for cut positions in our phasing.

Problem 6 (**Haplotype threading**). *Given a clustering $\mathcal{C}$ of reads and the position-wise relative coverages for each cluster, find a threading $\mathcal{T} = \left(T_{(1)},\ldots,T_{(m)}\right)$ with minimum total penalty according to the following penalty function*

$$\text{cost}_{total}(\mathcal{T}) = \underbrace{\sum_{i=1}^{m} \text{cost}_{cov}\left(T_{(i)},i\right)}_{\textit{coverage cost}} + \underbrace{\sum_{i=1}^{m-1} \text{cost}_{switch}\left(T_{(i)},T_{(i+1)}\right)}_{\textit{switch cost}}. \tag{3.15}$$

The best possible switch score would be achieved by binding each thread to some fixed cluster index, which could obviously lead to a very poor score for the coverage representation. In reverse, one could easily minimize the coverage penalties by assigning the expected cluster multiplicities at each variant for the price of losing all haplotype contiguity. Our goal was to tune the model such that it includes necessary switches (e.g. when the span of a cluster ends or its expected multiplicity changes) while accurately explaining the cluster coverages. The

choice of p_{switch} and p_{affine} on one hand and p_{cov} on the other hand controls which aspect the model prioritizes over the other.

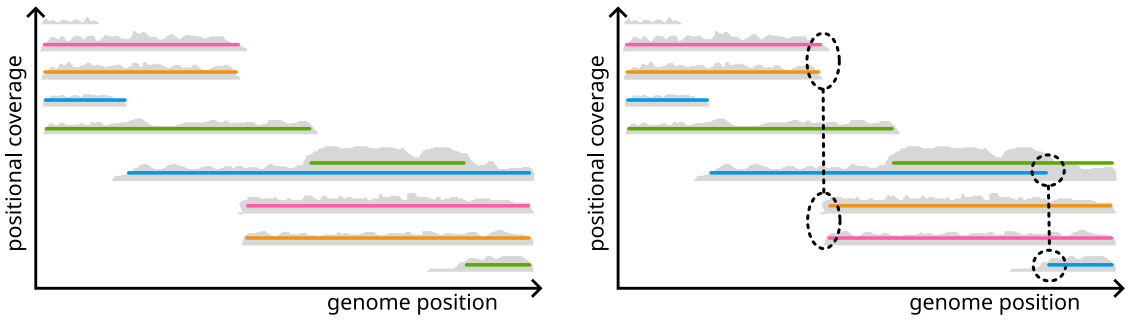
We realized that the variant density on the genome has a great influence on this choice. The lengths of clusters mostly depend on average read length (in terms of covered variants). For fixed p_{switch} and p_{cov} and a fixed average read length, the impact of cost_{cov} grows with higher variant density because multiplicity deviations are counted separately for each variant. Based on this observation, we use the average number of covered variants per read as a pivot and set p_{affine} to the next rounded-up integer, while setting p_{switch} to $4 \cdot p_{\text{affine}}$ and $p_{\text{cov}} = 1$. There is no specific calculation behind the exact ratios, but, empirically, they produced reasonable results and shifting the ratios slightly up and down (e.g. by a factor of 2) did not lead to significant changes in phasing performance. In the original publication, we did not use variant density as a pivot, which led to noticeably worse performance for variant densities differing much from those exhibited by the test datasets from development.

At this point, we want to note that the original threading model contained a third component named *genotype conformity*. In addition to the expected cluster multiplicities, one could also use the cluster consensi – the most abundant allele among all reads for each variant inside a cluster – to infer a genotype from a cluster tuple. If this genotype is not conform to the input genotypes for the current variant, the non-conform tuple would be pruned to speed up calculations. This worked great for the artificial polyploid human data with known gold-standard phasing (see Section 3.5.1) as the provided input genotypes steered the threading in the right direction. In the aftermath, we realized, though, that genotyping native polyploid organisms is more challenging and that genotype errors could even lead to correct solutions being replaced by incorrect ones. The most common problem was that genotype conformity as a hard constraint enforced switches on one thread, but on the next variant, where one would expect the evicted thread to revert to its old position, it was another thread taking its place. This led to a very jagged threading with many switch errors (see Figure 3.14 for an example). We thought about ways to circumvent this problem but finally decided to postpone genotype conformity to some post-processing step. We will further discuss this matter in Section 3.4.1.

3.3.3 Solving techniques

The dynamic program matches the one presented in [14]. For the new version of WHATSHAP POLYPHASE*, I added the concept of consensus lists.*

We developed a dynamic programming approach to rapidly find the optimal sequence of tuples that minimizes all costs. We compute a two-dimensional matrix S with one column for every variant position $1, \ldots, m$ and one row for every possible cluster tuple at v_i. Since the number of eligible cluster tuples can differ between variant positions, the columns of S do not necessarily have the same lengths. We denote the length of column i with l_i and the individual cluster tuples with $t_{1,i}, \ldots, t_{l_i,i}$. Using the cost functions defined above, $S(i, j)$ is then computed as



(a) Example of tetraploid threading (b) Critical switching positions

Figure 3.6: **Visualization of the threading.** (a) Clusters of reads are represented as grey shapes with their horizontal span indicating the covered variants and the height being the respective coverage. The $p = 4$ threads are shown as colored lines passing through the clusters. Multiple threads can co-enter the same cluster if the coverage is suited. (b) Alternative threading with the same score in our model. Two positions cause ambiguity and allow switches in the threading compared to (a). These are candidate cut positions to prevent switch errors in the final phasing.

$$S(j,1) = \text{cost}_{\text{cov}}\left(t_{j,1}, 1\right) \ , \tag{3.16}$$

$$S(j,i) = \text{cost}_{\text{cov}}\left(t_{j,i}, i\right) + \min_{k \in \{1,\ldots,l_{i-1}\}} \left(S(k, i-1) + \text{cost}_{\text{switch}}\left(t_{k,i-1}, t_{j,i}\right)\right) \text{ for } i > 1 \ . \tag{3.17}$$

The optimal threading score is then given by the minimum value in the last column. Starting at this position, we assemble the sequence of clusters with minimum costs via backtracking. The threading process is illustrated in Figure 3.6 for $p = 4$. The clusters from the first step are drawn as gray clouds in a two-dimensional space, as explained in the previous subsection. The position on the y-axis has no numerical meaning and is just used for illustration purposes. Starting from the left, a 4-tuple of the five present clusters needs to be chosen as the first cluster tuple. According to the coverage, the best choice is to thread one haplotype through each of the four clusters with the highest coverage and ignore the smallest one, as this probably collected only erroneous reads. From thereon, the threads change clusters whenever a cluster ends or undergoes a drastic change in relative coverage (like the green thread joining the blue thread).

From the optimal threading, we now derive a set of intermediate haplotypes $\tilde{H_1}, \ldots, \tilde{H_p}$. That is, we transform the cluster indices of each thread into a sequence of alleles. The most straightforward way would be to compute the consensus allele for each cluster and variant position and replace every cluster index with the corresponding allele. Problems arise from collapsed regions, where a cluster occurring twice in some cluster tuple should not necessarily be replaced by the same consensus allele twice, as the two represented haplotypes could contain different alleles for a few variant positions. We will consult the *allele depths* of each cluster instead and compute *consensus lists* for each cluster and variant to pick replacing alleles from.

Let $D_{C_j,i}(a)$ be the allele depth of allele a for cluster $C_j \in \mathcal{C}$ and variant v_i, i.e. the number of reads in C_j supporting a for variant position i. The corresponding consensus list is denoted as $L_{C_j,i}$. The first element in this list is always the most frequent allele among C_j for v_i. Each

consequent element is the allele with the highest absolute frequency divided by the number of occurrences in the list, up until the current position. If, for example, allele 0 occurs 30 times and allele 1 occurs 25 times, then allele 0 is picked for the first position. For the second position, we can either pick allele 0 a second time or allele 1 for the first time. Since $\frac{30}{2} < \frac{25}{1}$, the second element would be allele 1. The third element is again allele 0 because $\frac{30}{2} > \frac{25}{2}$. This is continued until all consensus lists have p elements to always provide enough alleles to pick from when replacing the cluster indices in our computed threading. If different alleles are to be selected for a cluster with multiplicity 2 or higher, it is not clear how to arrange these alleles on the different haplotypes. We will discuss this issue in Section 3.4.2 and for now, replace cluster indices with alleles from the allele lists in some arbitrary order.

3.3.4 Further optimizations

The following optimizations were not mentioned in [14], but the pre-selection of eligible cluster tuples existed before. I newly developed and implemented the symmetry elimination for this thesis.

For ploidy p the number of possible cluster tuples over l different clusters is l^p, rendering the threading model intractable if the selectable clusters are not restricted. In the original publication, only the $2 \cdot p$ clusters with the highest coverage were kept for the model on each variant position because any further clusters would drop below $\frac{1}{2p}$ in relative coverage, and thus always have an expected multiplicity of 0.

In the current version, this filter is more strict by (i) only keeping at most $p + 2$ clusters and (ii) always pruning clusters for a variant v_i, if it has less than $\frac{1}{8p}$ relative coverage on all variants in a small window (of size 10) around v_i. The first change was a compromise towards to otherwise unacceptably high running time for ploidies above 4, while the second change was an optimization for the average case, where the $p + 1$-th cluster (in descending coverage order) usually has a negligibly low coverage and does not significantly contribute to better solutions. Moreover, there exists a global row limit of 2^{p+4} for the DP for ploidies above 6, where only the 2^{p+4} best-scored entries are kept after computing each column.

All of the mentioned restrictions imply a loss of optimality in the original sense of the threading model, where there is no limit for the selectable clusters and rows in the DP table. The implementation rather poses a compromise between acceptable running time in practice and mathematical interpretability with minimal losses.

For practical implementation, we included elimination of symmetric solutions. Since haplotypes do not have a specific order, there is no need to enumerate all permutations of optimal haplotypes. Therefore, it is sufficient to only store the best representative of every group of permuted cluster tuples in each column. Two cluster tuples are permutations of each other if they contain the same multiplicities for every cluster but not necessarily the same order of cluster indices.

In every new column, we initially generate a canonical representative for every group of permuted tuples with its cluster indices in ascending order. For the first column, we can directly use this set as the final tuples because we only need to compute coverage costs, which do not depend on the order of the indices inside a tuple. For every candidate tuple in the following columns, however, we additionally need to find a predecessor tuple with minimal switch cost. These costs depend on the order of cluster indices inside the tuples; thus, it is not sufficient to just consider the canonical representatives.

The old implementation of WHATSHAP POLYPHASE solved this issue by generating all permutations and computing the switch costs for all pairs of tuples between the previous and the current column. To avoid these highly redundant computations, the new implementation uses a different routine for the switch cost, which considers all permutations of cluster tuples simultaneously. We temporarily generate canonical representatives for all tuples in the previous column (for the current one we already have them). Two canonical tuples T and T' are then compared as follows: Starting with $i = j = 1$, the routine checks whether $T[i] = T'[j]$. If yes, this counts as a match and both i and j are incremented by 1. If $T[i] > T'[j]$, only j is incremented and likewise, i is incremented if $T[i] < T'[j]$. The lowest number of switches between any permutation of T and any permutation of T' is then given by p minus the number of matches.

The described routine allows the new implementation of WHATSHAP POLYPHASE to efficiently find the optimal predecessor P for every canonical representative T in the current column. Finally, we permute T such that the switch cost between P and T is minimized and store the permutation as the final cluster tuple for the current column.

3.3.5 Breakpoints

The concept of breakpoints was already proposed in [14], but they were not named as such. I formalized breakpoints and collapsed regions for this thesis.

In Section 1.4.1 we introduced the concept of cut positions, a means to express discontinuity within the phasing output. The most obvious source of discontinuity is the absence of connecting reads for two consecutive variant positions. Even if the haplotypes could be resolved perfectly on both ends, it is impossible to link these sets of partial haplotypes correctly without additional long-range data. The shorter the reads and the lower the heterozygosity, the more fragmented the final phasing becomes due to this problem. Collapsed regions occurring exclusively in polyploid phasing are another obstacle to contiguity as these regions act like an absence of variation for a subset of haplotypes. In Figure 3.2 we already saw an illustration of this problem, where a region could not be uniquely phased, even with long, error-free reads.

Formally, we describe a collapsed region as a tuple $(v_{\text{first}}, v_{\text{last}}, c_{\text{id}}, T_{\text{coll}})$, where v_{first} and v_{last} are the first and last contained variant (inclusively), c_{id} is the associated read cluster index – for shortness, we will use the terms cluster and cluster index interchangeably in this

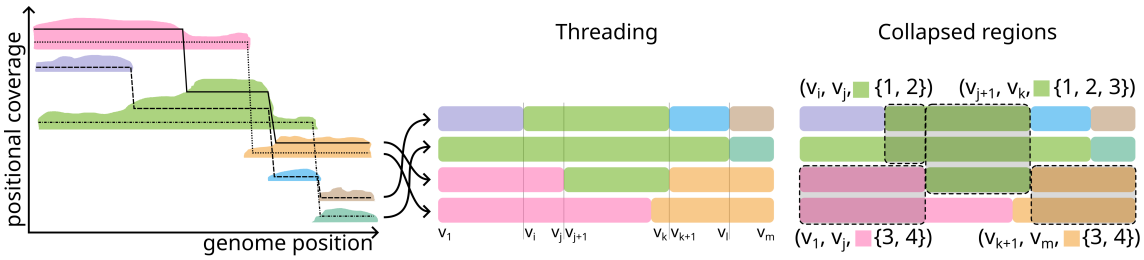


Figure 3.7: **Example for collapsed regions.** Left: The colored shapes represent read clusters. Their horizontal position and width determine their covered variants and their height shows their positional coverage. Four lines with different dash patterns are drawn over the clusters to represent the computed threading. Center: The four threads are shown here as colored bars to indicate on which cluster they reside for each of the variant positions. Right: Four collapsed regions are indicated by the gray transparent rectangles. Note that the green cluster contains two collapsed regions because of two different multiplicities (2 and 3). The magenta and the first green collapsed region share variants, but affect different threads.

section – and $T_{\text{coll}} \subseteq \{1, \ldots, p\}$ is the index set of affected threads in $\mathcal{T}$. By definition, the multiplicity of c_{id} inside $\mathcal{T}$ has to be at least 2. We additionally demand that the multiplicity of c_{id} has to be constant over the entire collapsed region and that the size of the collapsed region is maximal, i.e., it cannot be extended before v_{first} or beyond v_{last} without violating the constraint of constant multiplicity of c_{id}. In Figure 3.7, we see an example of four computed threads in the center and the corresponding collapsed regions on the right. Each thread is shown as a colored bar, where the colors indicate the position-wise cluster indices. Whenever a color is present on more than one thread for the same position, the latter must be part of a collapsed region. There are two collapsed regions containing the green cluster because the multiplicity of green is 2 from positions v_i to v_j and 3 from v_{j+1} to v_k. These two regions are neither allowed to be merged into one region nor are they allowed to be split because the latter would result in non-maximal collapsed regions. The definition allows collapsed regions to overlap in positions (like the magenta and the first green region), as long as they do not overlap in affected threads.

Finally, we introduce the concept of *breakpoints* to describe candidates for cut positions. Breakpoints are 2-tuples, consisting of a variant position – the one *before* the breakpoint – and a set of affected thread indices. Every time a collapsed region $(v_{\text{first}}, v_{\text{last}}, c_{\text{id}}, T_{\text{coll}})$ ends because of a *decrease* in multiplicity for c_{id}, we add the breakpoint $(v_{\text{last}}, T_{\text{coll}})$ to the set $\mathcal{B}$ of all breakpoints. On the contrary, we do not create a breakpoint if the multiplicity of c_{id} *increases* or when a collapsed region starts.

The reason why increases and decreases in multiplicities are treated differently, is that two or more threads only become indistinguishable after they have resided on the same cluster. If one of the threads switches to a different cluster, it is ambiguous which thread to pick from the perspective of our threading model. In Figure 3.7, the green cluster holds three threads at some point before two of them switch to the yellow and blue clusters because of a drop in coverage. However, any other configuration (e.g. moving the first thread to the yellow cluster and the third one to the blue cluster) would be equally good for the threading model, as it does not track the history of the threads. It just decides when a thread has to switch based on

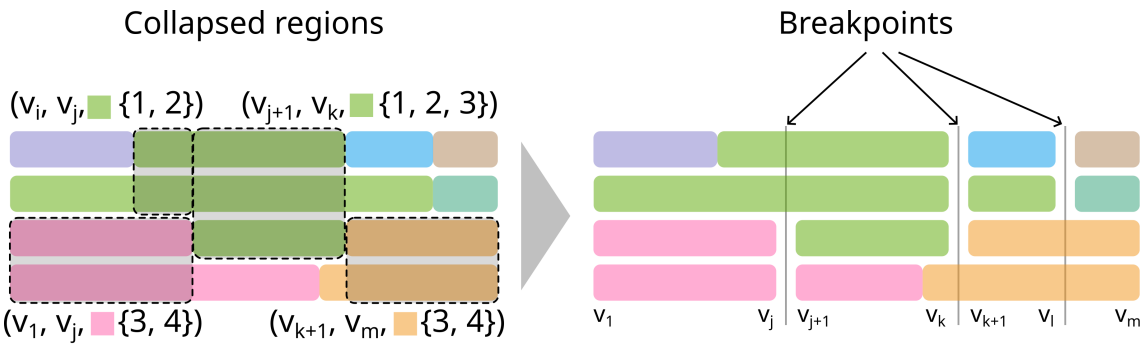


Figure 3.8: **Breakpoints.** Continues the example from Figure 3.7. Left: Collapsed regions marked on four threads with colors representing assigned clusters. Right: Inserted breakpoints according to the presented rules.

coverage penalties. If a thread enters a cluster that already holds a thread (i.e. the multiplicity of this cluster increases beyond 1), the history of the newly entering thread is still clear and we do not need to add a breakpoint.

In addition to collapsed regions, there is a more rare case, where the threading model faces ambiguity and forces us to define more breakpoints. Whenever two consecutive cluster tuples $T_{(i)}$ and $T_{(i+1)}$ are different, it is usually just one differing component and we can conclude that one cluster index was just replaced by another one, like for the purple cluster in Figure 3.7. However, if two or more components differ between $T_{(i)}$ and $T_{(i+1)}$ (e.g. the green and blue cluster switching to brown and cyan), the switch becomes ambiguous and we consider this position to be a breakpoint as well. We continue the example from Figure 3.7 in Figure 3.8 to show the resulting breakpoints from the previously determined collapsed regions. At the end of the magenta and the larger green region, we insert a breakpoint for the affected threads. We insert another breakpoint at the transition from the blue and green clusters to the brown and cyan clusters because more than one thread switches to a new cluster at the same position. All threads that are unaffected by the breakpoints are drawn as continuous stripes. The transition from the small green collapsed region to the larger one, for instance, is not affected by the breakpoint because the multiplicity of the green cluster increases.

In principle, the phasing pipeline is fully functional at this point: First, $\mathcal{T}$ defines p sequences of clusters, which can be translated into p sequences of alleles based on the clusters' consensi. Second, $\mathcal{B}$ can be considered as a set of cut positions, as they cover all sites of known ambiguity. However, there is still some remaining information we can use to further refine the current results: We can re-use the read information itself to act as a tiebreaker for some of the breakpoints and we can use the input genotypes explicitly to correct our predicted haplotype sequences for variants where the latter induce a different genotype (than the input ones). We will discuss these possibilities in the next section.

3.4 Refining results

Genotype conformity and cut position policies were present in the original algorithm [14] but this section as a whole has been newly written and includes different implementations of the two mentioned ideas.

As mentioned at the end of the previous section, the third stage of WHATSHAP POLYPHASE aims at refining the previously computed output. In the original publication, this stage was not presented as a dedicated step and also did not exist inside the code as such. In fact, the measures to find tiebreakers for ambiguities in the computed threading (explained in Section 3.3.5), which were used in the implementation then, were not even described in the original paper. Genotype conformity was originally enforced during the threading stage, but was later decoupled from it, as it caused artifacts in the phasing (see Section 3.6). This separation motivated us to summarize all refinement steps in their own stage. We recall again, that all follow-up work on the pipeline was never published, so this is the first documentation of the reordering stage. The stage itself is divided into three smaller steps: (i) enforcing the intermediate haplotypes to match the input genotypes for each variant, (ii) reordering the segments between breakpoints using read information, and (iii) selecting breakpoints as cut positions if the segments on each side could not be confidently linked via read information.

3.4.1 Genotype conformity

Genotypes are an optional input for haplotype phasing that provides a multiset of present alleles for each variant. Many phasing tools offer to explicitly use this information to refine their phasings, like H-POP-G [40], FLOPP [15], and the diploid WHATSHAP [12]. The original version of WHATSHAP POLYPHASE added genotype conformity as a hard constraint to the threading stage, i.e., cluster tuples could only be candidates if they resulted in the correct genotype.

We note here that the provided genotypes might be based on different or additional read data than what is given as input for the phasing problem itself. In case of different sequencing technologies, this might be a conscious choice: Short reads, for instance, are usually quite uninformative for phasing but their low error rate makes them very suitable to accurately estimate allele dosages of each variant site, assuming the short reads have been mapped correctly. Thus, following the given input genotypes can lead to more accurate results.

On the other hand, the intermediate haplotypes from the threading stage contain connectivity information of alleles over multiple variants that might not have been considered or available to the method used for creating the input genotypes. Figure 3.9 shows an example from an IGV [97] screenshot, where a local phasing could have prevented a wrong genotype call. The genotypes for positions 71,615,591 and 71,615,643 are both reported as `0/0/1/1` by the genotype caller GATK [37]. However, if we look at the reads below (gray horizontal stripes), we can see that the alternative alleles on the first position (red rectangles) are only present on a subset of reads that contain the alternative allele on the second position (blue

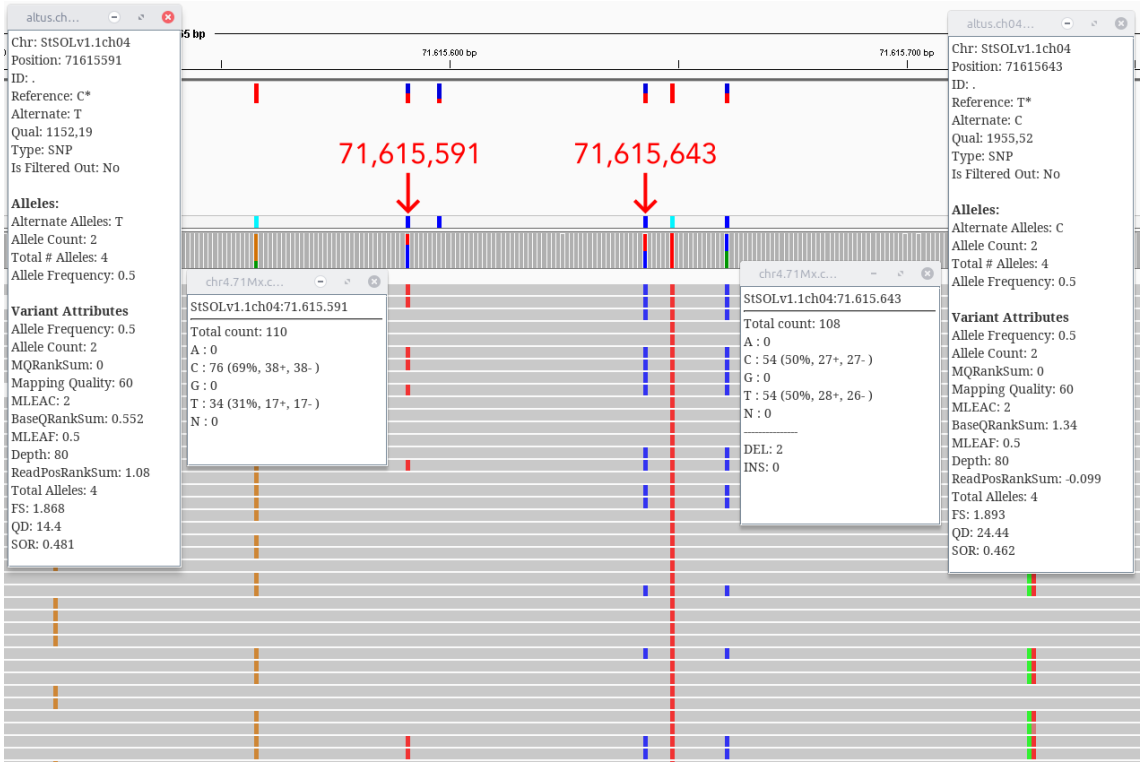


Figure 3.9: **Genotyping error.** Example screenshot from IGV, showing the genotype calls for two positions 71,615,591 and 71,615,643, (marked by red arrows. The vertical bars below the arrows the allele distribution among the reads. Each of the gray stripes below represents one read and each deviation from the reference allele is marked by some color (red for T, blue for C).

rectangles). This implies that the second must have more alternative alleles, which makes `0/0/0/1` a more likely choice for the first position.

Since the quality of the input genotypes can greatly vary, the question of whether they should be preferred over the induced genotypes from the intermediate haplotypes cannot be answered in general. From practical experience, the phased organism also plays an important role. While genotype callings on human data (diploid) turned out to be very accurate during testing, called genotypes on potato samples (tetraploid) seemed to contain more inconsistencies, especially when mixing different datasets from different sequencing technologies.

The new version of WHATSHAP POLYPHASE performs the genotype enforcement independently for each variant. We therefore describe the process from the perspective of a single variant v_i. Let $t = (c_1, \ldots c_p)$ be the tuple of cluster indices computed by the threading algorithm with consensi alleles $a_1, \ldots, a_p \in \{0, \ldots, l-1\}$ and induced genotype G_i'. Finally, let G_i be the input genotype for v_i. As a reminder, the absolute frequency of allele a in genotype G is defined as $f_a(G)$.

If $G_i = G_i'$, all alleles already occur in the requested dosage and the phasing is left untouched. Otherwise, some excess alleles have to be replaced with underrepresented ones. Let $A^+ := \{a \mid f_a(G_i') > f_a(G_i)\}$ and $A^- := \{a \mid f_a(G_i') < f_a(G_i)\}$ be the set of alleles with excess and shortage among the intermediate haplotypes, respectively. Furthermore, let $A := A^+ \cup A^-$ be the union of both sets and $I := \{i \in \{1, \ldots, p\} \mid a_i \in A^+\}$ be the haplotype indices containing an

excess allele (for variant v_i). For the redistribution, we take each allele $a \in A^+$ exactly $f_a(G_i)$ times (i.e. the required amount of times) and each $a \in A^-$ exactly $f_a(G_i) - f_a(G_i')$ times (i.e. the number of missing occurrences) and insert them in a multiset A_{place}. Note that $|A_{\text{place}}| = |I|$ because the number of required occurrences of alleles $a \in A^+$ is represented by both sets, while the number of missing allele occurrences (accounted for in A_{place}) must be equal to the number of excess occurrences (accounted for in I). To balance out the allele occurrences we take all unique permutations over the alleles in A_{place} and assign them to the haplotype indices in I, which gives us several alternatives for how to redistribute alleles in accordance with G_i.

To determine the best allele assignment of A_{place} on haplotype indices I, we look at the allele depths of the clusters with an index in I. We determine the multiplicity of each allele for each cluster given by the selected allele assignment and compute the likelihood of the observed allele depths given the allele multiplicities for each cluster. This yields one likelihood value for each possible allele assignment and we pick the one with the highest likelihood. As already explained, this process is repeated for each variant independently.

3.4.2 Resolving collapsed regions

We previously omitted the task of resolving heterozygous variants inside collapsed regions. Based on the consensus lists from Section 3.3 (or on the just-discussed genotype enforcement), the haplotypes inside a collapsed region may be assigned differing alleles for some of the variants, even though a collapsed region is associated with a single read cluster. This is not a contradiction to the idea of collapsed regions because the affected haplotypes might not be completely identical; the differences were rather too small to separate the contained reads in the presence of other, much more divergent haplotypes. The way we aim to resolve heterozygosity in collapsed regions is to view each region as a new phasing instance: The ploidy of a collapsed region is at last 2 – otherwise it would not be collapsed –, it contains a distinct set of heterozygous variants, and the associated read cluster provides the relevant subset of reads to resolve the collapsed region. We make use of this observation and use WHATSHAP POLYPHASE recursively to resolve each collapsed region independently. This idea was already illustrated in Figure 3.3 by the selection box over the two green threads in the bottom right with almost identical sequences. In that example, the sub-instance has ploidy 2 and two heterozygous variants, highlighted in red.

After solving a sub-instance, we use the returned haplotype sequences to arrange just the heterozygous positions inside the associated collapsed region. If the sub-instance yields any new breakpoints, we add these to the running set of breakpoints accordingly because if two variants inside the sub-instance cannot not be confidently connected, we also lack the confidence to connect the heterozygous variants inside the collapsed region. If a collapsed region contains all haplotypes, its sub-instance would have the same ploidy as the full instance and we omit the recursive call to avoid the danger of a non-terminating recursion.

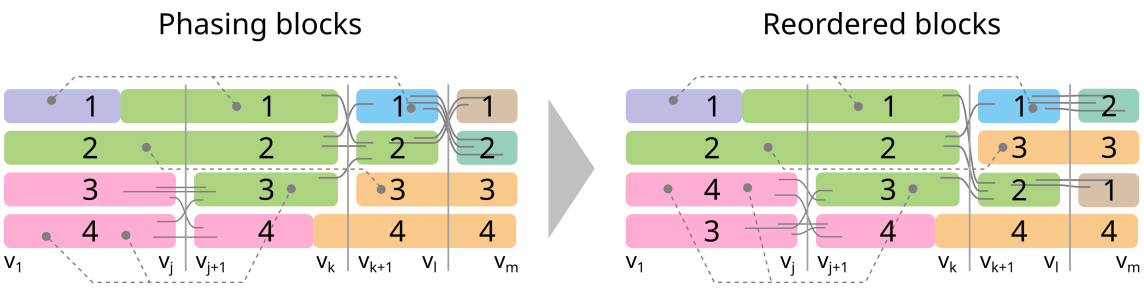


Figure 3.10: **Illustration of phase block reordering.** We continue the example from Figure 3.8. Left: For each of the four phasing blocks, the haplotype blocks are indexed from 1 to 4. The gray filled lines indicate reads that align to haplotype blocks from different phasing blocks. The dashed lines indicate connections between certain alleles on different haplotype blocks provided by a pre-phasing. Right: The haplotype blocks are reordered with respect to the linkage information.

The recursive solving was not done in the old algorithm, as differences inside collapsed regions were handled by the cluster refinement. This design was slightly contradictory to the choice of cluster editing as a base model that intentionally allows for collapsed regions.

3.4.3 Reordering phase blocks

After all collapsed regions are internally resolved, we call the interval between two consecutive breakpoints a *phasing block* and each of the p individual allele sequences therein a *haplotype block*. From now on, we consider all of the haplotype blocks to be fixed and only optimize the order of haplotype blocks within the same phasing block. For this purpose, we use two inputs: (i) the read sequences themselves and (ii) optionally a pre-phasing from the input VCF.

The purpose of reordering is to identify haplotype blocks from neighboring phasing blocks that we can uniquely link with sufficient confidence provided by the reads or a pre-phasing. If all haplotype blocks between two phasing blocks can be linked (i.e. we have evidence of which pairs of haplotype blocks belong to the same haplotype), we can discard the intermediate breakpoint later on to obtain larger phasing blocks with the p contained sequences aligned accordingly.

The reordering process is shown in Figure 3.10, for which we again continue the example from Figure 3.8. We index the haplotype blocks within each phasing block to track their order. The gray lines connecting different tips of the threads represent reads from the affected clusters (colors represent clusters) and their endpoints indicate to which haplotype blocks they fit best. We remark that the span of the read clusters is not aligned to their occurrences in the threading. The blue cluster, for instance, is only present on a small interval of the threading, but the contained reads might span more variants in front or behind this interval. Thus, it is possible to match the reads from the blue cluster to the brown or green haplotype blocks in the neighboring phasing blocks.

The dashed lines in Figure 3.8 indicate the aforementioned pre-phasing. As a reminder, a pre-phasing is a set of phased but incomplete haplotypes that is given as an optional input. In this example, it connects haplotypes among several phasing blocks. With reads and pre-phasing combined, we can reorder the haplotype blocks as shown on the right side of this

figure. The reads are not helpful in resolving the first and second breakpoints but give a clear hint that the brown and cyan haplotype blocks should swap their position in relation to the previous phasing block. In most applications, no pre-phasing is available and many breakpoints remain unsolved. We still consider pre-phasings for our reordering model, though, because our second polyploid phasing method of this thesis (see Chapter 4) produces sparse phasings that we can leverage as additional information for WHATSHAP POLYPHASE.

Let l be the number of phasing blocks and $\mathcal{B} = \{(\text{pos}_1, T_1^{\text{aff}}), \ldots, (\text{pos}_{l-1}, T_{l-1}^{\text{aff}})\}$ be the set of breakpoints with their respective positions and sets of affected threads. For a breakpoint $\mathcal{B}_b$ affecting $|T_b^{\text{aff}}| \leq p$ many haplotypes, there are $|T_b^{\text{aff}}|!$ many arrangements between the neighboring blocks. We assume that these arrangements are enumerated in some canonical way and that we can retrieve a score $l_{b,i}$, expressing the likelihood of observing the given reads from the affected clusters when applying the i-th arrangement over $\mathcal{B}_b$. We only compute such read scores for transitions between neighboring blocks, even though reads could theoretically cover more multiple breakpoints. For the calculation, we generate all $|T_b^{\text{aff}}|!$ arrangements and compute the likelihood for each read to be sampled from any of the arranged haplotype-block pairs with a low allele error rate of 0.7. The final score for an arrangement is the logarithm of the product of likelihoods for all reads in the affected clusters.

A pre-phasing $\overline{H}_1, \ldots, \overline{H}_p$ provides linkage information over arbitrarily large distances. With $a_{b,h,t}$ we denote a similarity score between the t-th haplotype block in phasing block b and $\overline{H}_h$. In case no pre-phasing is given, all $a_{b,h,t}$ can be set to some arbitrary number, say 0. The block reordering problem can be characterized as a combinatorial optimization problem with the arrangement within each block being the "choices" to make and support among reads and pre-phasing defining the objective to maximize.

Problem 7 (**Phasing block reordering**). *Given a set of breakpoints $\mathcal{B}$ inducing l phasing blocks and a pre-phasing $\overline{H}$. Let Π be the set of all permutations over $\{1, \ldots, p\}$ and $i : \Pi \to \{1, \ldots, p!\}$ be some canonical mapping for each permutation to a unique index. Find permutations $\pi_1, \ldots, \pi_l \in \Pi$ for the l phasing blocks that maximize the following expression:*

$$\max_{\pi_1,\ldots,\pi_l \in \Pi} \underbrace{\sum_{b=1}^{l-1} l_{b,i(\pi_b^{-1} \circ \pi_{b+1})}}_{\textit{linkage likelihoods}} + \underbrace{\sum_{b=1}^{l} \sum_{h=1}^{p} a_{b,h,\pi_b(h)}}_{\textit{pre-phasing likelihoods}}, \tag{3.18}$$

where $l_{b,i}$ is the likelihood for the i-th canonical permutation over breakpoint b based on read information and $a_{b,h,t}$ is the likelihood that the t-th haplotype block of the b-th phasing block belongs to the h-th haplotype of $\overline{H}$. The $\circ$-operator denotes the sequential application of two permutations and $\pi(h)$ is the h-th element of permutation π.

We decided to set up an integer linear program to solve the general case of having both read and pre-phasing information. Without pre-phasing, the blocks can be re-arranged one after another, allowing for a polynomial-time algorithm with respect to the number of blocks l, the number of variants m, and the maximum number $p!$ of arrangements per block.

The ILP uses the following variables:

$$x_{b,t,h} := [\![\text{haplotype block } t \text{ is placed on haplotype } h \text{ for block } b]\!]$$
$$y_{b',t_1,t_2} := [\![\text{haplotype blocks } t_1 \text{ from } b\text{-th phasing block and } t_2 \text{ from } (b+1)\text{-th phasing block are on the same haplotype}]\!]$$
$$z_{b',i} := [\![i\text{-th permutation is used to connect haplotype blocks over breakpoint } b']\!]$$

with $b \in \{1,\dots,l\}$, $b' \in \{1,\dots,l-1\}$, $i \in \{1,\dots,|T_{b'}^{\text{aff}}|\}$ and $h,t,t_1,t_2 \in \{1,\dots,p\}$.

While the x-variables model the final assignment of haplotype blocks onto haplotypes, the y-variables describe which individual haplotype blocks from two neighboring phasing blocks reside on the same haplotype in the chosen assignment. Lastly, the z-variables state the indices of all chosen arrangements between neighboring phasing blocks. These indices follow the same canonical enumeration as used for computing the scores $l_{b,i}$ for read support and $a_{b,h,t}$ for the pre-phasing affiliations. The objective function contains one component for each type of score:

$$\max \sum_{b=1}^{l-1} \sum_{i=1}^{|T_b^{\text{aff}}|!} z_{b,i} l_{b,i} + \sum_{b=1}^{l} \sum_{t=1}^{p} \sum_{h=1}^{p} x_{b,t,h} a_{b,t,h} \tag{3.19}$$

First, we ensure a one-to-one mapping between haplotype block and haplotype for each block:

$$\sum_{h=1}^{p} x_{b,t,h} = 1 \qquad \forall b \in \{1,\dots,l\}, \forall t \in \{1,\dots,p\} \tag{3.20}$$

$$\sum_{t=1}^{p} x_{b,t,h} = 1 \qquad \forall b \in \{1,\dots,l\}, \forall h \in \{1,\dots,p\} \tag{3.21}$$

In case no pre-phasing is given, we can eliminate some symmetric solutions by fixing the haplotype blocks inside the first phasing block and only reorder the other phasing blocks, i.e. setting $x_{0,t,t}$ to 1 for every $t \in \{1,\dots,p\}$. This is legal since haplotypes do not have a specific order. In the same manner as the haplotype-block-to-haplotype assignments, haplotype blocks are linked to exactly one haplotype block in the next phasing block:

$$\sum_{t_1=1}^{p} y_{b,t_1,t_2} = 1 \qquad \forall b \in \{1,\dots,l-1\}, \forall t_2 \in \{1,\dots,p\} \tag{3.22}$$

$$\sum_{t_2=1}^{p} y_{b,t_1,t_2} = 1 \qquad \forall b \in \{1,\dots,l-1\}, \forall t_1 \in \{1,\dots,p\} \tag{3.23}$$

For each two consecutive phasing blocks b and $b+1$, we only allow those haplotype blocks to be reordered that are affected by the b-th breakpoint in between. All unaffected haplotype blocks are just linked to their neighbors with the same index:

$$y_{b,t,t} = 1 \qquad \forall b \in \{1,\dots,l-1\}, \forall t \notin T_b \tag{3.24}$$

When combined, Equations 3.22, 3.23, and 3.24 only allow haplotype block indices in T_b to be reordered. The following constraint enforces consistent variable assignments between x- and y-variables, by forcing haplotype blocks t_1 and t_2 to be linked into consecutive blocks if t_1 and t_2 are assigned to the same haplotype.

$$x_{b,h,t_1} + x_{b+1,h,t_2} - y_{b,t_1,t_2} \leq 1 \qquad \forall b \in \{1,\dots,l-1\}, \forall h, t_1, t_2 \in \{1,\dots,p\} \tag{3.25}$$

Finally, we model the logic of the z-variables by setting $z_{b,i}$ to 1 if and only if the haplotype block linkage over breakpoint b follows the i-th arrangement over this breakpoint, given as a function $\text{perm}_i : T_b^{\text{aff}} \to T_b^{\text{aff}}$:

$$z_{b,i} \geq \sum_{t=1}^{p} y_{b,t,\text{perm}_i(t)} - |T_b^{\text{aff}}| + 1 \qquad \forall b \in \{1,\dots,l-1\}, \forall i \in \{1,\dots,|T_b^{\text{aff}}|\} \tag{3.26}$$

$$z_{b,i} \leq y_{b,t,\text{perm}_i(t)} \forall b \in \{1,\dots,l-1\}, \forall i \in \{1,\dots,|T_b^{\text{aff}}|\} \forall t \in \{1,\dots,p\} \tag{3.27}$$

After solving the integer linear program, we are given a haplotype assignment for each of the haplotype blocks in each phasing block. Rearranging these segments in the specified way yields the final haplotype sequences.

3.4.4 Detecting cut positions

With the haplotype blocks reordered, the last remaining task is the selection of cut positions. As stated at the end of Section 3.3, the breakpoints $\mathcal{B}$ already pose a suitable choice of cut positions. In the reordering step, however, we gathered several likelihood scores for different haplotype-block arrangements, from which we can derive confidence values for each breakpoint regarding our selected reordering. Since we not only want to minimize phasing errors but also maximize block lengths, we should only select breakpoints as cut positions if they cannot not be resolved with high confidence. We can freely choose the corresponding confidence threshold and thereby control the balance between block length and block correctness, which is a key feature of WHATSHAP POLYPHASE.

As stated before, for each breakpoint b and each possible permutation i of affected threads, we computed a likelihood $l_{b,i}$ expressing how well the rearrangement of threads explains the observed reads overlapping the breakpoint. Let i' be the actually chosen permutation. The *confidence* of breakpoint b is then defined as $\frac{l_{b,i'}}{\sum_i l_{b,i}}$. Ideally, the confidence is close to 1, if all but the chosen permutation have almost no support among the reads.

Starting from the first block with a confidence of 1 for each haplotype, we multiply the confidence of each passed breakpoint onto the confidences of all haplotypes affected by the

sensitivity	q	r	policy in old implementation
5	1	1	any thread switches
4	0.99	2	every breakpoint
3	0.5	2	two threads switch simultaneously
2	0.5	$\min(3, p)$	two variants connected by too few reads
1*	0	p	two variants completely unconnected
0*	0	p	never

Table 3.1: Block cut thresholds for different sensitivity levels. For sensitivity level 1, breakpoints never introduce cut positions but the input is split over completely unconnected reads, making the new scheme identical to the old one. The same holds for sensitivity level 0, which completely avoids any cuts.

breakpoint. If the aggregated confidence of a haplotype drops below some threshold q with $0 < q \leq 1$, we mark this haplotype as unreliable. Once at least r haplotypes became unreliable, we mark the last seen breakpoint as a cut position, reset the aggregated confidences to 1 again, and continue the procedure.

Historically, WHATSHAP POLYPHASE has an integer parameter called *block cut sensitivity*, ranging from 0 to 5. It was used to specify which type of breakpoint should lead to a cut position in the final phasing. We replaced this qualitative method with the above scheme, as some breakpoints might be resolvable well enough to avoid a cut position without introducing errors. Table 3.1 shows the associated thresholds q and r for each sensitivity level and the old equivalent. The default sensitivity level is 4.

3.5 Experiments

This section re-used content from [14]. The setup of experiments in Section 3.5.1 is identical to the mentioned article. The other datasets and evaluation of the revised WHATSHAP POLYPHASE *as well as* FLOPP *have been newly added.*

To evaluate the performance of our developed algorithm, we conducted several benchmarks on multiple datasets. Like in the original publication of WHATSHAP POLYPHASE, we compare the results to H-POP-G, another state-of-the-art polyploid phasing tool. We also included the tool FLOPP, which was released in 2022 and was shown by Shaw and Yu to offer superior performance compared to H-POP-G and the first version of WHATSHAP POLYPHASE. As benchmarks, we first recreated the experiments from our original publication [14]. Second, we used the data generation routine from Shaw and Yu [15] to create similar test instances as in the respective paper. Finally, we used some real sequencing data from a potato cultivar for a set of small genomic regions.

We ran all algorithms with default settings and compared the resulting phasings to the ground truth haplotypes. It became obvious that different algorithms use different strategies on where to cut the phasing. For instance, H-POP-G defines phased blocks based on the connected components of the underlying reads by introducing cuts between pairs of variants

that are not connected by any sequencing reads. FLOPP does not divide the phasing at all, but reports the entire chromosome as a single phasing block. As explained in Section 3.4.4, WHATSHAP POLYPHASE uses a more sensitive approach, typically leading to much shorter but more accurate phasing blocks. To make the results more comparable, we also ran WHATSHAP POLYPHASE without inserting any cut positions unless there is no connection at all between two consecutive variants (denoted as WH-PP*). This is achieved by an additional parameter `-B 1` when running the tool from the command line. This strategy should be comparable to the cut strategy of H-POP-G.

To highlight improvements on or changes to the original WHATSHAP POLYPHASE, we included the old v1.0 version, which is the first release after the publish date, and compared it to the most recent release by the time we conducted the experiments, namely version v2.2. In some cases, we disabled the genotype enforcement by running our tool with the `--distrust-genotypes` parameter. For clarity, we use the abbreviations from Table 3.2 for different versions and configurations of WHATSHAP POLYPHASE.

abbreviation	version	additional parameters
WH-PP	v2.2	
WH-PP*	v2.2	`-B 1`
WH-PP^{d}	v2.2	`--distrust-genotypes`
WH-PP*^{d}	v2.2	`-B 1 --distrust-genotypes`
WH-PP$_{\text{OLD}}$	v1.0	
WH-PP*$_{\text{OLD}}$	v1.0	`-B 1`

Table 3.2: Abbreviations for different versions and configurations of WHATSHAP POLYPHASE.

Whenever available, WHATSHAP POLYPHASE was run with the `--reference` parameter because it is able to determine read alleles on variants more accurately by realigning reads to a given reference genome. We used the same machine and workflows to execute the experiments as in the previous chapter.

3.5.1 Artificial polyploid human data

The experiments were run as in [14]. The data generation pipeline was created by Jana Ebler.

We generated a tetraploid, pentaploid, and hexaploid version of human Chromosome 1 by combining sequencing data of three individuals (NA19240, HG00514, and HG00733), for which high-quality trio-based haplotype information is available [48]. We refer to these haplotypes as ground truth haplotypes. We merged PacBio sequencing data for the first two samples to produce tetraploid data at coverages $40X$ and $80X$. For hexaploid datasets, we added PacBio data from the third sample, again generating coverages 40× and 80×. Using the read simulator PBSIM [98], we additionally generated equivalent simulated tetraploid, pentaploid, and hexaploid data sets with the same coverages and known read origin. The pentaploid datasets

coverage	method	SFR (%)	SER (%)	HR (%)	N50 (bp)
40×	WH-PP	0.83	1.01	3.75	33455
	WH-PP$_{\text{OLD}}$	0.40	0.47	1.40	35649
	WH-PP*	1.49	1.77	28.94	2108824
	WH-PP*$_{\text{OLD}}$	1.04	1.19	27.87	2108824
	H-PoP-G	1.36	1.78	28.35	2195085
	FLOPP	4.49	2.60	39.85	248898070
80×	WH-PP	0.57	0.69	3.26	46003
	WH-PP$_{\text{OLD}}$	0.28	0.32	1.45	54467
	WH-PP*	1.09	1.28	28.97	2417376
	WH-PP*$_{\text{OLD}}$	0.74	0.83	27.69	2587104
	H-PoP-G	0.99	1.24	27.70	2587104
	FLOPP	5.63	2.07	40.31	248898070

(a) real tetraploid read data

coverage	method	SFR (%)	SER (%)	HR (%)	N50 (bp)
40×	WH-PP	0.71	0.88	3.34	39393
	WH-PP$_{\text{OLD}}$	0.37	0.44	1.76	49778
	WH-PP*	1.27	1.53	27.84	1908549
	WH-PP*$_{\text{OLD}}$	0.88	1.00	26.02	1908549
	H-PoP-G	1.25	1.69	26.62	2017098
	FLOPP	4.16	2.18	39.19	248898070
80×	WH-PP	0.31	0.35	2.29	53055
	WH-PP$_{\text{OLD}}$	0.27	0.29	2.94	81829
	WH-PP*	0.75	0.84	26.57	2172622
	WH-PP*$_{\text{OLD}}$	0.64	0.70	25.61	2172622
	H-PoP-G	0.82	1.02	25.71	2153145
	FLOPP	5.51	1.70	40.11	248898070

(b) simulated tetraploid read data

Table 3.3: Comparison of WHATSHAP POLYPHASE, H-PoP-G and FLOPP on tetraploid real (a) and simulated (b) datasets. Performances are based on the switch flip rate (SFR), switch error rate (SER), block-wise Hamming rate (HR) and N50 for the block size. The total length of the chromosome is 249 Mb.

were created by first generating hexaploid datasets with 20% higher coverage (i.e. 48× and 96×) and then filtering out all reads belonging to the second haplotype of sample HG00733.

Tetraploid data

The results for the tetraploid datasets are summarized in Table 3.3. In terms of N50 block size, there is a clear separation between WH-PP and WH-PP$_{\text{OLD}}$ with less than 100kb on the one hand and WH-PP*, WH-PP*$_{\text{OLD}}$, and H-PoP-G with about 2Mb and more on the other hand. Likewise, the Hamming rate is in the low single-digit range for the first two configurations and around 25–30% for the other three. FLOPP is the only algorithm to not output multiple blocks, resulting in an N50 block size of 250Mb and a Hamming rate of about 40%.

Throughout all tests, the switch flip rate (SFR) is always slightly lower than the switch error rate (SER), except for FLOPP which exhibits a substantially higher SFR than the other

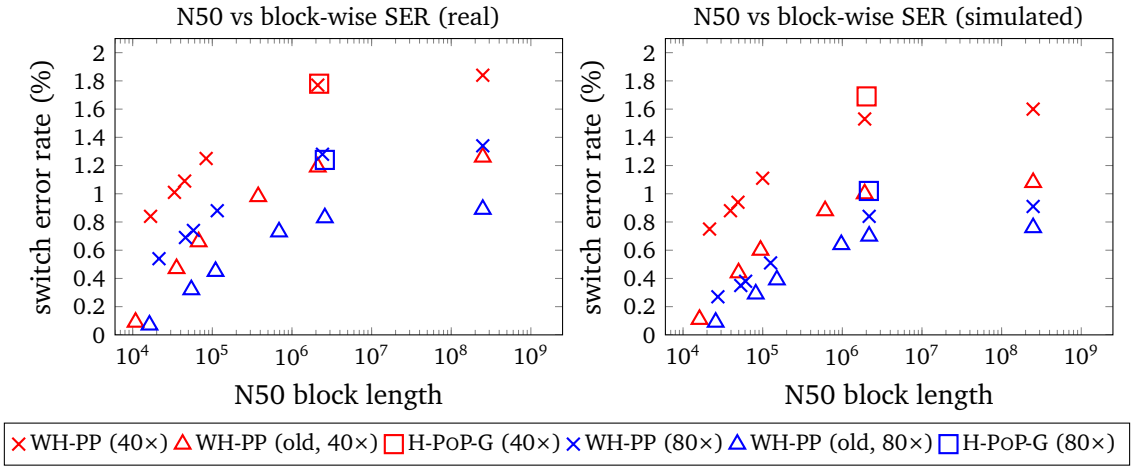


Figure 3.11: **N50 block lengths vs lock-wise switch error rates.** N50 block lengths and the respective block-wise switch error rates for different block cut strategies of WHATSHAP POLYPHASE on the real tetraploid read dataset (top) and the simulated tetraploid dataset (bottom) with 40× and 80× coverage.

methods. The former behavior is to be expected because using flips and switches to transform a computed phasing into the true one should at most require as many operations as using switches only. The only exception is if the computed phasing contains many wrong genotypes, which we will investigate in a separate test.

Compared to H-PoP-G, WH-PP$^*_{\text{OLD}}$ achieves about 30% lower switch flip rates on comparable N50 block sizes, matching the findings in [14]. With smaller blocks, the switch flip rates are about three times lower than for H-PoP-G, again confirming the published results. Since we re-sampled and re-simulated the reads, the exact numbers deviate slightly from the ones in the article. Interestingly, the new version of WHATSHAP POLYPHASE falls slightly behind H-PoP-G in terms of SFR and SER, so we can observe a performance regression compared to the old version.

Including the HR again, as well as the configurations with short phasing blocks, we can even see that the old version of WHATSHAP POLYPHASE outperforms the new one in every performed test. This is also visible in another reproduced experiment from the original publication, where we plot the N50 block sizes against the switch flip rate for all available block cut strategies of WHATSHAP POLYPHASE (see Figure 3.11). Since we have two versions of the algorithm, there are two sets of data points per dataset. As a comparison, we included H-PoP-G as well in the plot.

Each method is represented by a different shape, while the color stands for the coverage. Comparing again H-PoP-G and WH-PP$_{\text{OLD}}$, we see that H-PoP-G is above the Pareto-curve of WH-PP$_{\text{OLD}}$ (triangle-shaped dots) for the same coverage, as was reported in the article. A comparison between the triangle-shaped and the x-shaped dots reveals that the old algorithm indeed outperforms the new one by a noticeable margin. For the simulated reads, the new algorithm almost catches up when using the full 80× coverage. We also plotted the N50 block size against the Hamming rate (see Supplementary Figure B.1). It shows the same tendencies

cut strategy	version	refinement	SFR (%)	SER (%)	HR (%)	N50 (bp)
`-B 1`	new	yes	0.97	1.12	28.46	2587104
	old	yes	0.74	0.83	27.69	2587104
	new	no	1.09	1.28	28.97	2417376
	old	no	1.07	1.30	29.38	2587104
`-B 4`	new	yes	0.51	0.62	3.76	61928
	old	yes	0.28	0.32	1.45	54467
	new	no	0.57	0.69	3.26	46003
	old	no	0.44	0.58	1.10	35355

(a) real tetraploid read data

cut strategy	version	refinement	SFR (%)	SER (%)	HR (%)	N50 (bp)
`-B 1`	new	yes	0.68	0.76	25.93	2172622
	old	yes	0.64	0.70	25.61	2172622
	new	no	0.75	0.84	26.57	2172622
	old	no	1.00	1.20	27.50	2172622
`-B 4`	new	yes	0.34	0.37	3.47	90118
	old	yes	0.27	0.29	2.94	81829
	new	no	0.31	0.35	2.29	53055
	old	no	0.40	0.52	1.30	43364

(b) simulated tetraploid read data

Table 3.4: Comparison of WHATSHAP POLYPHASE with and without cluster refinement on tetraploid datasets with coverage 80×.

as the switch flip rate, although the curves of both WHATSHAP POLYPHASE versions are closer together.

Special test: Cluster refinements

The performance regression in WH-PP must be linked to the various changes in the algorithm. Since the code structure changed substantially between the two versions, it was not feasible to replace single modules to trace back the regressions. However, the cluster refinement of the old implementation could be easily ported to the new one. During the development of the initial algorithm, this feature turned out to reduce the number of switch errors by a noticeable amount and was always run by default. The backport is not available in a regular version of WHATSHAP but only accessible from an experimental branch that is based on version v2.2. The instructions how to retrieve the applied state of the code can be found in Appendix D.

We reran the tetraploid benchmarks for the new algorithm with cluster refinement enabled and for the old one with cluster refinement disabled, to see if this feature explains the discrepancies. Table 3.4 summarizes the results for coverage 80×. Enabling the refinement benefits the new algorithm in almost every case, but the old one stays ahead in all tests with only the N50 block size being slightly better for the new algorithm and short phasing blocks. However, without cluster refinement old algorithm's performance deteriorates significantly in switch flip

coverage	method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants	N50 (bp)
40×	WH-PP*	1.49	1.77	28.94	0	245798	2108824
	WH-PP*^{d}	2.81	0.99	29.62	7.28	240066	2108824
	FLOPP	4.49	2.60	39.85	8.34	223931	248898070
80×	WH-PP*	1.09	1.28	28.97	0	246583	2417376
	WH-PP*^{d}	1.97	0.78	29.64	4.76	242455	2587104
	FLOPP	5.63	2.07	40.31	12.13	231668	248898070

(a) real tetraploid read data

coverage	method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants	N50 (bp)
40×	WH-PP*	1.27	1.53	27.84	0	245727	1908549
	WH-PP*^{d}	2.60	0.84	28.94	7.10	240173	1908549
	FLOPP	4.16	2.18	39.19	8.62	223256	248898070
80×	WH-PP*	0.75	0.84	26.57	0	246846	2172622
	WH-PP*^{d}	1.36	0.65	26.60	2.87	244520	2170280
	FLOPP	5.51	1.70	40.11	12.19	233031	248898070

(b) simulated tetraploid read data

Table 3.5: Comparison of WHATSHAP POLYPHASE and FLOPP, with and without trusting genotypes.

rate and N50 block size. While it still outperforms the new algorithm on the real dataset, it falls behind on the simulated dataset.

As a conclusion, the refinement is an integral part of the old algorithm to boost phasing quality. The new algorithm only benefits slightly, as it possesses other mechanisms to deal with collapsed regions. As we did not replace or re-implement other parts, the cause of the performance regressions remains unknown as of now.

Special test: Distrust genotypes

In Table 3.3 we saw that FLOPP reached significantly higher switch flip rates than the other methods and was also the only method, where this rate was higher than the switch error rate. Further investigation revealed that FLOPP is the only method to diverge from the input genotypes, even though the tested parameters (`-v`) are supposed to make the phasing follow the input genotypes, according to the documentation[1]. We also noticed that it was missing a noticeable fraction of variants compared to the output of other methods. The new version of WHATSHAP POLYPHASE is able to diverge from input genotypes as well when run with the flag `--distrust-genotypes`. We repeated the experiment from before, but this time also reported the fraction of wrong genotypes in the phasing, as well as the number of phased variants.

As we can see in Table 3.5, the behavior of WHATSHAP POLYPHASE follows that of FLOPP when allowing genotype divergence. The switch flip rate goes up and surpasses the switch error rate. The explanation is that in order to transform the computed haplotypes into the

[1] *https://github.com/bluenote-1577/flopp*

true ones, all genotype deviations have to be repaired using flips. For instance, if 7% of all phased variants contain a wrong genotype, then for a tetraploid sample, this causes at least $\frac{0.07}{4} = 1.75\%$ of all reported alleles to be wrong as well. Indeed, subtracting a quarter of the wrong genotype rate from the switch flip rate brings the latter very close to or slightly below the respective switch error rate.

We observe a slight drop in phased variant count of about 2% for WH-PP^{*d} and a noticeable drop of 6 to 10% for FLOPP. There are several reasons why a phaser might decide to not phase a specific variant. Since the only difference between WH-PP* and WH-PP^{*d} is the genotype handling, the missing variants are likely variants that have been wrongly predicted to be homozygous by the latter. The evaluation routine only considers common heterozygous variants between the output and the ground truth VCF files, as homozygous variants are always considered to be unphased. This explanation probably also holds true for FLOPP, although the genotyping gets worse for 80× coverage (as opposed to WH-PP^{*d} and the expectation that more coverage makes genotyping more accurate), while the variant count increases from 40× to 80×.

Interestingly, the genotype divergence causes the switch error rate for WH-PP^{*d} to drop below the level of the old implementation. By definition, the switch error rate can only be computed for variants with conforming genotypes. Thus, variants with wrong genotypes are excluded from this metric. We conclude that these variants are more erroneous on WHATSHAP POLYPHASE and there is room for improvement regarding genotype preservation. This observation might also be related to the general performance regression of the new algorithm, as genotype conformity is handled in a completely different way compared to before.

Hexaploid data

The hexaploid datasets behave similarly to the tetraploid ones (see Table 3.6). The Hamming rate, switch error rate, and N50 block size are close to the results from [14]. That is, H-POP-G shows 40 to 60% higher SER than the WH-PP$^{*}_{\text{OLD}}$. The new implementation falls even further behind the old one for hexaploid data, reaching almost twice the SFR and SER for long blocks and a bit more than twice the error rates for short blocks. A quick test using the `--distrust-genotypes` parameter resulted in a significantly lower SER, on par with the old implementation (see Supplementary Table B.2). This further confirms the hypothesis of genotype enforcement being a significant source of errors. As expected, phasing quality generally deteriorates with increased ploidy for all tested methods. This is due to lower haploid coverage and an exponentially larger solution space.

Pentaploid data

The simulated pentaploid data (see Supplementary Table B.1) offers no novel observations and was mostly done for the sake of reproducing old results. Interestingly, the phasing quality

coverage	method	SFR (%)	SER (%)	HR (%)	N50 (bp)
40×	WH-PP	1.98	2.43	3.60	8758
	WH-PP$_{\text{OLD}}$	0.84	1.00	1.67	16028
	WH-PP*	3.42	4.10	27.79	4265325
	WH-PP*$_{\text{OLD}}$	1.96	2.27	27.30	4265325
	H-PoP-G	2.77	3.66	26.99	4846407
	FLOPP	6.28	3.63	32.86	248889649
80×	WH-PP	1.27	1.52	2.83	15162
	WH-PP$_{\text{OLD}}$	0.48	0.53	1.14	25429
	WH-PP*	2.30	2.70	27.94	6412111
	WH-PP*$_{\text{OLD}}$	1.27	1.41	25.68	6412111
	H-PoP-G	1.80	2.29	26.68	6412111
	FLOPP	7.65	2.58	34.20	248898070

(a) real hexaploid read data

coverage	method	SFR (%)	SER (%)	HR (%)	N50 (bp)
40×	WH-PP	2.07	2.64	3.45	8947
	WH-PP$_{\text{OLD}}$	0.88	1.06	1.83	16882
	WH-PP*	3.43	4.24	27.52	3754960
	WH-PP*$_{\text{OLD}}$	1.99	2.31	26.96	3754960
	H-PoP-G	2.79	3.85	26.10	3847617
	FLOPP	6.85	3.48	33.08	248848964
80×	WH-PP	1.03	1.25	2.12	14656
	WH-PP$_{\text{OLD}}$	0.43	0.48	0.96	26251
	WH-PP*	2.04	2.40	26.55	4490129
	WH-PP*$_{\text{OLD}}$	1.23	1.36	26.08	4607645
	H-PoP-G	1.80	2.31	25.84	4945599
	FLOPP	7.52	2.46	34.15	248848964

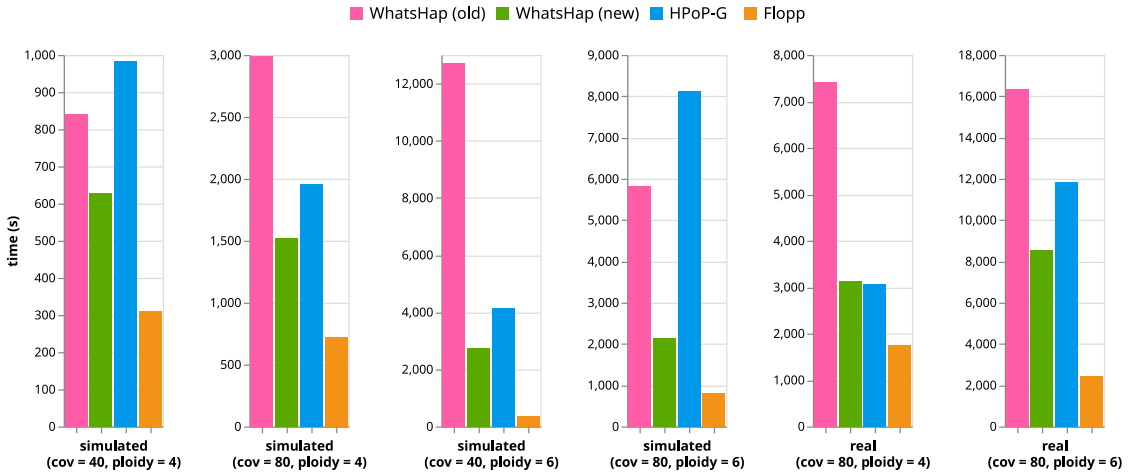
(b) simulated hexaploid read data

Table 3.6: Comparison of WHATSHAP POLYPHASE, H-PoP-G and FLOPP on hexaploid real (a) and simulated (b) datasets. Performances are based on the switch flip rate (SFR), switch error rate (SER), block-wise Hamming rate (HR) and N50 for the block size. The total length of the chromosome is 249 Mb.

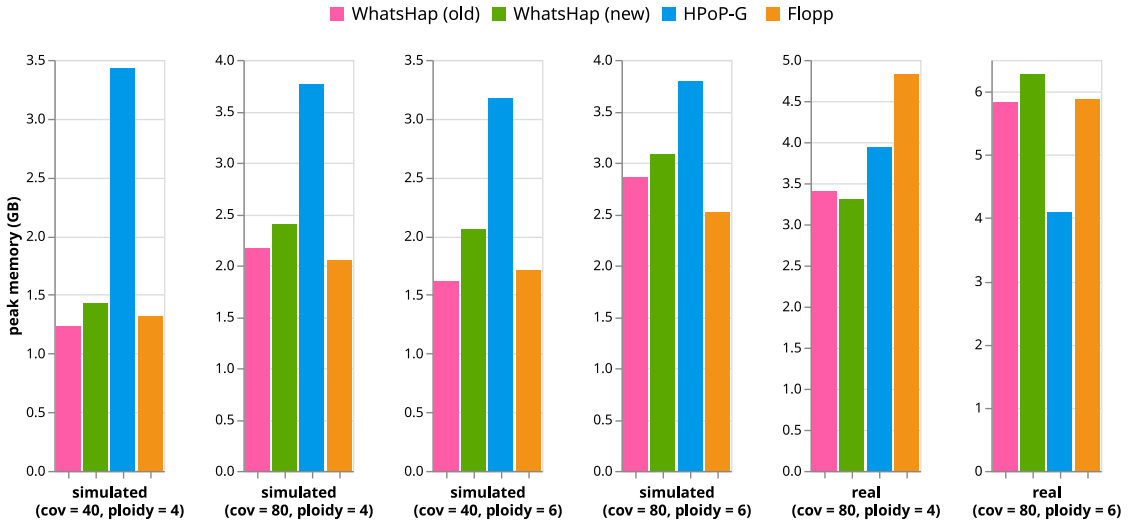
was overall even lower than for the hexaploid data, which contradicts the previous results, where the pentaploid numbers ended up in between the tetraploid and hexaploid ones.

Runtime and memory consumption

Another aspect of implemented tools is the required computational resources. To measure the runtime, we repeated a selection of tests but made sure that at most two instances were running simultaneously on our machine with multithreading disabled if a tool had this option. This ensured that the CPU was constantly operating at its highest frequency (3400 Mhz) and reduced competing memory and disk accesses to a reasonable minimum. We used the `time`-command with flag `-v` to measure the determine the total runtime and reported the "Maximum resident set size" as peak memory consumption for each run.



(a) Runtime in seconds for different datasets.



(b) Peak memory consumption in GiB for the same datasets.

Figure 3.12: **Runtime and memory consumption.** Each bar plot represents one dataset, which is given below the x-axis. The top row (a) shows the runtime in seconds for each algorithm. The bottom row (b) shows the peak memory consumption in GiB. WHATSHAP POLYPHASE was run with block cut sensitivity `-B 1`.

Figure 3.12 summarizes the resources used by each algorithm. For the simulated reads we tested two ploidy levels (4 and 6) as well as both coverage levels (40× and 80×). For the real reads we only used the high-coverage sets to outline differences to the simulated reads. We ran WHATSHAP POLYPHASE with block cut sensitivity `-B 1` because these settings are closer to the competing algorithms H-PoP-G and FLOPP. Using the default block cut sensitivity resulted in only 5–10% lower runtimes but a noticeably lower memory footprint because the readset is split more aggressively, and thus the algorithm has to keep less data in memory at a time during execution.

We see that the new version of WHATSHAP POLYPHASE is substantially faster than the old one in out tests. On the tetraploid datasets, this is mainly caused by a more optimized implementation of the cluster editing solver and the avoidance of cluster refinement iterations. On the hexaploid datasets, the threading stage gains a great speedup as well due to improved symmetry elimination. A more detailed runtime profile for both implementations is given in

Supplementary Figure B.2. Interestingly, the slowest step for the new version is the input processing.

The fastest algorithm in all instances is FLOPP. While it takes about half the time of WHATSHAP POLYPHASE on tetraploid data, the speedup on hexaploid data is up to 7. Comparing absolute numbers, we can see that WHATSHAP POLYPHASE takes significantly more time to phase higher ploidies, while FLOPP scales better for this parameter. H-POP-G is about 40% slower than WHATSHAP POLYPHASE on average.

The memory consumption of WHATSHAP POLYPHASE slightly grew for the newer version by about 15%. On the simulated datasets, H-POP-G has the largest memory footprint by a fair margin, while FLOPP uses slightly less memory than our algorithm. For the real data, the ordering is rather chaotic. In general, all algorithms are quite memory-efficient and could have executed all conducted experiments on a regular up-to-date PC or laptop.

3.5.2 Simulated Solanum tuberosum data

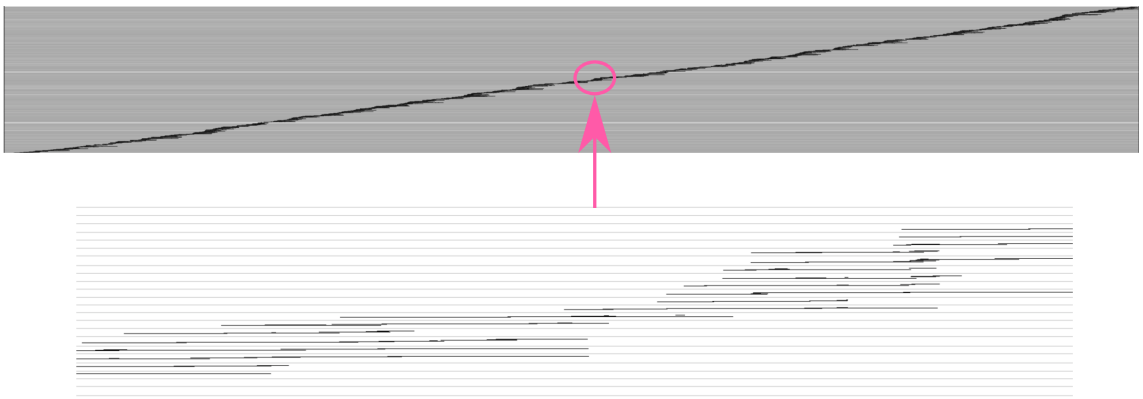
The article by Shaw and Yu [15] – introducing FLOPP as a novel phasing method – contained a variety of benchmarks, where FLOPP was compared to H-POP-G and the old version of WHATSHAP POLYPHASE. While FLOPP was shown to outperform the other tools in the selected benchmarks, WHATSHAP POLYPHASE severely struggled on the tested datasets. We were able to reproduce these issues with a small triploid test dataset provided on the author's GitHub repository[2].

Test dataset

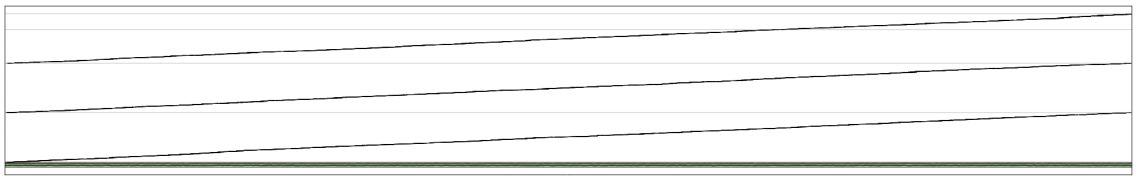
To get an idea of where the old WHATSHAP POLYPHASE fails and to exclude any implementation errors, we analyzed all intermediate steps. The read clustering turned out to be very fragmented as illustrated in Figure 3.13a. Each cluster only contained a tiny amount of reads, which makes it impossible for the downstream steps to produce a coherent phasing. The culprit turned out to be the clustering refinement because the reads were almost perfectly separated when disabling this feature.

This did not resolve all issues, however, as the computed threading based on the high-quality clustering still contained lots of switch errors. Since one could almost trivially assign the threads to the four clusters, this behavior was unexpected. A visualization of the computed threading revealed many undesired thread switches, shown in Figure 3.14. Even though the three visible clusters (gray horizontal shapes) are present over the entire viewed window of variants and the coverage is reasonably split among them, threads tend to switch to another cluster for a single variant and back again. In some cases, this only impacted the result minimally, e.g. for the red arrow, marked with a "1", in other cases the switching thread did not

[2] *https://github.com/bluenote-1577/flopp*



(a) Read clusters with refinement.



(b) Read clusters without refinement.

Figure 3.13: **Visualization of the read clusters.** Each read is represented as a black horizontal bar, where the horizontal reach stands for the covered variants. Clusters of reads are separated by gray horizontal lines. All clusters are stacked vertically. (a) Shows the clustering of original WHATSHAP POLYPHASE with default parameters. The lower part is a zoomed-in picture of the red circle in the upper part. (b) Shows the clustering when running the same algorithm without cluster refinement.

return to its original cluster but instead switched places with another thread, e.g. for the red arrow, marked with a "2". These "jumps" are initiated by the genotype constraints of the old threading model because genotype deviations are completely forbidden. This forces the threading model to choose expensive adjustments against the intuition of the whole model. Since the genotype conformity cannot be disabled in the old version of WHATSHAP POLYPHASE, this issue cannot be fixed by a different choice of command line parameters.

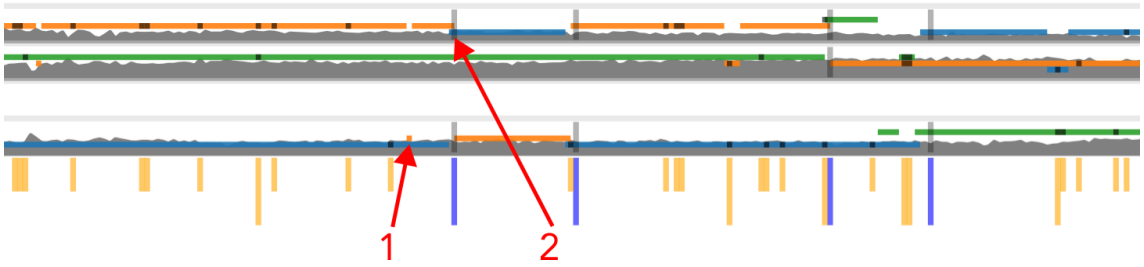


Figure 3.14: **Visualization of the haplotype threading.** This figure shows a small section of a computed haplotype threading by the original WHATSHAP POLYPHASE algorithm. Each horizontal shape (in this case three) represents a cluster. The height of the gray shape indicates the read coverage of the corresponding cluster for each variant position. The orange, green and blue lines represent the optimal threading. The vertical bars below indicate flip errors (yellow) and switch errors (dark blue). Vertical gray bars mark affected haplotypes for a switch errors, black dots indicate haplotypes affected by flip errors.

Nevertheless, we ran all algorithms on the test dataset and summarized the results in Table 3.7. Indeed, the old WHATSHAP POLYPHASE implementation offers very poor phasing quality for both long and short phasing blocks. H-POP-G and the WH-PP are very close in

method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants	N50 (bp)
WH-PP	0.48	0.77	0.52	0	67190	3009993
WH-PP$_{\text{OLD}}$	7.84	9.64	9.14	0	39083	279
WH-PP*	0.50	0.80	0.54	0	67291	3019741
WH-PP$^{*}_{\text{OLD}}$	11.83	15.18	44.28	0	49726	3018718
H-PoP-G	0.46	0.74	0.46	0	67378	3019643
FLOPP	0.97	0.01	0.97	2.88	64236	3016025

Table 3.7: Comparison between WHATSHAP POLYPHASE, H-PoP-G and FLOPP on simulated Solanum tuberosum data, provided as test data by Shaw and Yu.

every evaluated metric, showing that the changes offer a great improvement over the old algorithm. Switch flip rates and Hamming rates are about twice as high for FLOPP compared to the previous two methods, but the switch error rate is excellent. This means that FLOPP is able to almost perfectly phase this dataset, except for the 2.88% of positions containing wrong genotypes.

The issues found with this kind of data triggered further algorithmic engineering on WHATSHAP POLYPHASE and led to the revised version presented in this thesis. The new version of WHATSHAP POLYPHASE does not run into these issues because the clustering is not followed by a refinement step and the updated threading model is oblivious to genotypes and cluster consensi.

Newly simulated data

For further validation of our findings, we used the instructions provided by the supplementary GitHub repository[3] of FLOPP to reproduce some of the performed benchmarks in [15]. These are based on the first 3.5Mb on the first chromosome of the v4.04 assembly of S. tuberosum [99]. The authors used the HaploSim tool from Motazedi et al. [59] to generate haplotypes for ploidies 3, 4, 5, and 6, the read simulator PaSS [100] to simulate PacBio reads and NanoSim [101] for Oxford Nanopore reads with 10, 15, and 20× coverage per haplotype. We limited the scope of our tests to PacBio reads and the lowest level of coverage because these settings seem to be closest to the supplied test dataset.

In contrast to the test data, the used reference sequence is now available as well. For the old WHATSHAP POLYPHASE algorithm this makes a surprisingly huge difference, as we can see for the triploid experiments in Table 3.8a. The switch flip and switch error rates improved by a factor of 50 compared to the triploid test data, while the new implementation also gained an improvement of about factor 5 in these two metrics. Since H-PoP-G and FLOPP do not use the reference sequence, there was no significant deviation from the numbers of the test dataset.

This puts WHATSHAP POLYPHASE in front of H-PoP-G regarding the two metrics, also for the hexaploid experiments shown in Table 3.8b. FLOPP shows excellent switch error rates for

[3] *https://github.com/bluenote-1577/flopp_test*

method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants	N50 (bp)
WH-PP*	0.08	0.13	9.47	0	67469	3019845
WH-PP*^{d}	0.26	0.01	0.26	0.75	63707	3017396
WH-PP*$_{\text{OLD}}$	0.36	0.50	39.82	0	67020	3018904
H-PoP-G	0.41	0.69	0.43	0	67657	3019580
FLOPP	0.95	0.01	0.95	2.82	64545	3016166

(a) simulated triploid PacBio reads

method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants	N50 (bp)
WH-PP*	0.21	0.34	24.99	0	66729	3020108
WH-PP*^{d}	0.70	0.02	14.83	4.06	66171	3019140
WH-PP*$_{\text{OLD}}$	0.33	0.44	32.95	0	66764	3019399
H-PoP-G	0.40	0.69	0.40	0	67026	3019941
FLOPP	1.11	0.02	3.01	5.68	64606	3016166

(b) simulated hexaploid PacBio reads

Table 3.8: Comparison between WHATSHAP POLYPHASE, H-PoP-G and FLOPP on simulated Solanum tuberosum data.

the price of having about 3 to 6% wrongly genotyped variants. As a comparison, we again included the setting WH-PP*^{d}. This puts WHATSHAP POLYPHASE on the same level as FLOPP in terms of switch error rate, with a noticeable advantage in switch flip rate due to fewer genotyping errors.

The Hamming rate turns out to be the weak spot for WHATSHAP POLYPHASE in all settings. For the test data, WH-PP* achieved competitive Hamming rates, as well as WH-PP*^{d} for the newly simulated triploid data. This proves that WHATSHAP POLYPHASE is generally able to produce coherent phasings, but the few remaining switch errors are often in critical spots, where large parts of the predicted phasing block get switched. These observations also hold for tetraploid and pentaploid experiments (see Supplementary Table B.4).

Since H-PoP-G and FLOPP are able to phase all the datasets in one large block without critical switches, we omitted the short block settings for WHATSHAP POLYPHASE. Despite most of the problems being solved by the availability of a reference genome, the new implementation outperforms the old one in every aspect for this set of experiments. In Supplementary Figure B.3, we added measures for runtime and memory consumption.

3.5.3 High-confidence regions from Altus cultivar

In Chapter 4, we will present another polyploid phasing method that combines sequencing data with genotype information for a large offspring panel from a pair of parental plant samples. The evaluation of this method was based on sequencing and genotype data for a potato cultivar, named “Altus”. We use the HiFi sequencing data from Altus with about 96× coverage to benchmark the methods discussed in the current chapter. The variant calling and genotyping were done using high-depth short-read data, and might thus be incongruent with

method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants
WH-PP*	6.94	0.49	21.83	23.84	6654
WH-PP$^*_{\text{OLD}}$	8.27	1.81	37.77	23.86	6732
WH-PP^{*d}	1.84	0.12	8.36	6.91	6509
FLOPP	0.69	0.03	22.24	2.47	6567

Table 3.9: Benchmarks for Hifi reads from Altus cultivar on the region ch04:71,586,000-71,947,000.

the ground truth data obtained from a HiFi assembly. For further details about the data, see Section 4.3.

Since the tested regions were quite short, we omitted the short block configurations of WHATSHAP POLYPHASE. Also, H-POP-G was unable to process the input files because the VCF contains several multi-allelic variants. We again included the genotype-oblivious setting WH-PP^{*d}, as the genotyping was not done on different data than the input reads in this case. From Table 3.9 and Supplementary Table B.5, we can see that genotype divergence is very high for all tests. WH-PP* performs consistently better than WH-PP$^*_{\text{OLD}}$, with more than 50% fewer switch errors. The switch flip rate is mostly dominated by the wrong genotypes, but still lower for the new implementation.

The genotype-oblivious methods WH-PP^{*d} and FLOPP again show a very small switch error rate, but also a much lower fraction of wrong genotypes. Over all three regions, FLOPP is able to predict the correct genotype from the input reads much better than WHATSHAP POLYPHASE. This leads to about 50% fewer genotype errors and proportionally lower switch flip rates. Regarding the Hamming rate, none of the methods show convincing results here with WH-PP^{*d} taking the lead in two out of three regions. The runtime and memory consumption tests in Supplementary Figure B.3 show little differences between old and new versions of WHATSHAP POLYPHASE. As in previous tests, FLOPP is about twice as fast as WH-PP* (and WH-PP^{*d}) but this time also uses significantly less memory. The reason for the latter is not clear to us, as FLOPP has been on par with our algorithm on larger datasets regarding memory consumption. We did not investigate any further why this behavior changes for smaller datasets.

3.6 Discussion

Throughout this chapter, we presented WHATSHAP POLYPHASE, a novel approach for polyploid haplotype phasing. Its design choices are based on observations regarding the arising challenges for generalizing the phasing problem to polyploid genomes. Compared to the original version of WHATSHAP POLYPHASE, we presented a revised version with new ideas and described the underlying methods in much more detail than the original publication. While the intention of the new read pair scoring scheme was mostly to have a more rigorous model with fewer assumptions and a clearer objective, the extraction of genotype conformity from the threading model was rather motivated by the reproduced issues reported by Shaw and Yu.

We compared both the old and new versions of WHATSHAP POLYPHASE to other state-of-the-art tools, namely H-POP-G and FLOPP, on three different kinds of data. To our surprise, the old algorithm produced considerably better results than the new one on the artificial polyploid human data, especially for higher ploidies. On the contrary, the new algorithm consistently outperformed the old one on the simulated Solanum tuberosum data and on the selected genome regions of the Altus cultivar. One explanation could be overfitting because, for the original publication, we only tested the first version of WHATSHAP POLYPHASE on the artificial polyploid human dataset and tuned all parameters with this data in mind. The cluster refinement improved phasing quality remarkably on this dataset but also inhibited a proper result for the test dataset from [15], where no reference genome was provided. Our conclusion is that the algorithmic changes represent an improvement to our method, as it became more robust to a wider variety of data.

It is quite difficult to determine a clear winner among the tested methods, as all of them exhibit different strengths. H-POP-G shows consistent performance over all tests, except for the lack of support for multi-allelic variants. FLOPP is very good at minimizing the number of switch errors, although it drastically fell behind the other methods on the polyploid human data. When combining the different configurations of WHATSHAP POLYPHASE (i.e. always take the best-performing one), they produce the lowest switch flip and switch error rate. However, these few switch errors are often critical, resulting in bad global haplotype reconstructions compared to the other methods. Especially the results of FLOPP and H-POP-G in Section 3.5.2 for higher ploidies show, that the input data allows for continuous and correct phasings. One advantage of WHATSHAP POLYPHASE is the configurable block cut strategy, which no other method offers, to the best of our knowledge.

3.6.1 Limitations

Looking at the experimental results from a broad angle, the overall phasing quality of the presented methods is still far away from what one usually expects in the diploid field. While this is mostly attributed to the computational complexity of polyploid phasing, there are two concerns arising from the data itself. We discussed the challenge of collapsed regions and the practice of cutting phasings into blocks to communicate the uncertainty here. WHATSHAP POLYPHASE tackles this by applying different cut strategies based on the computed threading. The results from Section 3.5.1 show that reasonably low block-wise Hamming rates induce very small blocks, which limits the usability of the output. Since WHATSHAP POLYPHASE showed susceptibility to critical switch errors in Sections 3.5.2 and 3.5.3, it is unclear whether this is an algorithmic weakness or whether the long-read sequencing data in general is not sufficient to provide better results for complex genomes.

The second concern is the limitation of linear reference genomes. Polyploid phasing grew from the existing research field of diploid phasing, where the assumption of such a reference genome is reasonable. Plant genomes, however, tend to contain much more structural varia-

tion, and are thus very hard to be projected onto a linear reference. This can lead to wrong or fragmented read alignments, wrong genotyping, and other artifacts. For instance, if one haplotype contains a large deletion, this effectively induces a local drop in ploidy that is not trivial to detect and requires awareness within genotyping and phasing tools. In Section 3.5.3 we observed a huge discrepancy between called genotypes of two different read data sets in the same genomic region. Such issues are usually not well represented in simulation studies, as the respective tools rarely include structural variation.

3.6.2 Ideas for future work

Following up on the example of local ploidy drops, support for dynamic ploidy could be an interesting feature in practice. The first challenge would be to predict a local ploidy, based on read coverage and alignment artifacts, like large non-reference sequences between two aligned read segments. This is a bit out of scope for a haplotype phaser and could rather be treated as a separate problem. The second challenge is to incorporate such local ploidy information into the phasing model of WHATSHAP POLYPHASE. The major changes would be located in the threading stage, where the model explicitly computes a fixed amount of intermediate haplotype sequences. We had the idea to introduce a residual cluster that is supposed to "absorb" threads on regions with reduced ploidy. The model would still compute p sequences but with the additional constraint of using the residual cluster for unnecessary threads. However, this idea was never implemented, mostly due to the lack of appropriate data for benchmarking such a feature.

Moving further away from a linear reference, one might even think of graph-based alignments as input for a phasing algorithm. Each read then covers a sequence of nodes in a reference graph instead of a sequence of alleles. The basis of our read clustering model is the ability to measure the similarity of two reads. With a proper similarity model, cluster editing could be applied without adjustments to the underlying alignment structure. For haplotype threading, however, we would again need some definition of where the front and back of the chromosome are located in the graph and which nodes are considered to be co-linear to each other. Thus, an extension of WHATSHAP POLYPHASE to graph-based alignments would only be possible on graphs with a linear coordinate system, where all nodes are traversed in forward order with respect to the coordinates.

Within the bounds of linear reference genomes, the experiments showed potential for improvement on genotype conformity. Allowing WHATSHAP POLYPHASE to deviate from the input genotype resulted in much lower switch error rates and negated the observed regressions of the new algorithm in this metric. This implies that variants, where the WHATSHAP POLYPHASE disagrees with input genotypes, are hotspots for errors and should receive more attention in algorithmic improvements.

Chapter 4

WhatsHap Polyphase Genetic

This chapter is based on [16], which was published in iScience. I was involved in most of the steps but was the main contributor to the algorithmic concept and its implementation. For this thesis, I added more details to the methodology and the context of the method.

In Chapter 2, we saw how reference-based diploid phasing can be enhanced by combining sequencing data from related individuals and applying Mendel's law. This raises the question, of whether this also holds for polyploid phasing and how the additional information can be embedded into existing approaches. In the following, we will revisit our previous work, describing a phasing method for datasets with two autopolyploid parental samples and a large population of progeny samples that were bred from the parental ones, also called F1 population [16]. First, we will discuss possible challenges of generalizing the concepts from diploid pedigree phasing to polyploid genomes. Second, we will present the developed method and discuss its applicability and its relationship to pure read-based methods, like WHATSHAP POLYPHASE.

4.1 Heredity in polyploid genomes

In the diploid reproduction process, we assume both copies of each chromosome to be homologous, i.e. sharing the same overall structure and most of the present genes. In polyploid genetics, however, we have to differentiate between allopolyploidy and autopolyploidy. Autopolyploid genomes are formed from genome duplications, resulting in more than two homologous chromosomal copies per cell. In case of an even ploidy $p = 2q$, the offspring inherits q chromosomal copies from each of its parents. A prominent example of this category is the potato (Solanum tuberosum), from which most cultivars are known to be autotetraploid [34, 102].

On the opposite, allopolyploid genomes occur after hybridization of different taxa, possibly from different species. As a result, the chromosomal copies are diverged from each other and their differentiation on a sequence level becomes easier. In some cases, the genome can be separated into its diploid progenitor species – and thus be assembled or phased like a diploid

genome – as has been shown for wheat [103], strawberry [104, 105], and peanut [106]. Since duplication and hybridization events can occur independently from each other, some species have accumulated both traits during evolution. One example is the hexaploid sweet potato, consisting of a diploid and a tetraploid progenitor genome [9].

What sets the hereditary process apart from the already visited diploid case is the huge number of possible transmissions. At each locus in an autotetraploid genome, there are $\binom{4}{2} = 6$ possibilities to select two haplotypes from each of the parents, resulting in a total of $6 \cdot 6 = 36$ possible transmissions. Compared to the four possibilities given in diploid species, it shows that tracking all combinations, like in [13], becomes intractable very quickly.

4.1.1 Previous work

The inclusion of pedigree information into polyploid phasing has been explored by previously existing tools. TriPoly [42] follows a similar idea as HapTree [60], where a statistical model is used to infer the most likely set of haplotypes, given the observed input reads. Since the number of haplotypes grows exponentially with the number of heterozygous variants, both methods implement a heuristic that constructs the haplotypes by one variant at a time. TriPoly extends the model to support two parental samples and an arbitrary number of shared offspring samples. Similar to the findings of Garg et al. [13], the pedigree-based model outperforms other tested polyploid phasers, which only support a single sample [42].

In Section 3.1, we already discussed that estimating the right dosage of alleles is much harder in polyploid genomes and usually requires high coverages. As deep sequencing with long reads is costly, there is a demand for methods working on short-read data. However, even considering the high SNP density of many plant genomes, short reads still provide very sparse connectivity and usually result in very short phasing blocks. For plant species, one way to overcome this problem is to breed many offspring samples from a single pair of parents. Since every part of a parental autopolyploid genome is, on average, transmitted to half of its descendants, correctly phased parent sequences should be shared by about half of the bred population. Recombinations lead to chimeric sequences among the offspring, but can in most cases be considered rare enough to not skew this relationship.

The research was continued by many of the authors of TriPoly and resulted in a follow-up tool, named PopPoly [43]. Instead of sequencing a single or few trios with with long reads, PopPoly shifts the focus towards larger F1 populations, sequenced with short-read technology. In their experimental study, the authors used population sizes of up to 30 samples and were able to achieve superior phasing performance over existing approaches, including TriPoly, with populations of five or more offspring samples.

4.2 WhatsHap Polyphase Genetic

We pick up the idea of a large offspring population with low sequencing depth and present a novel approach to use the population data for haplotype phasing. As opposed to the existing method, we do not construct haplotype sequences directly from reads but rather use the genotype information of parents and offspring population to identify so-called *marker alleles* within the parent samples and which of them reside on the same haplotypes.

More technically, we designed our algorithm for the following scenario: We are given genotype data for two parental input samples s' and s'', and allele depths for q progeny samples $s_1, \ldots, s_q$. The aim is to phase one of the parents, say s'. The other parent s'' can be phased in the same way, by switching the roles of s' and s'' and re-running the algorithm. We intentionally do not aim at phasing the progeny because in our setting only low-depth sequencing data is available for these samples.

In contrast to the previous definitions, let m here be the number of bi-allelic variants that are heterozygous on s'; all other variants are omitted for now. For a single variant, we call the more frequent allele among s' and s'' the *majority* allele and the other one the *minority* allele. Without loss of generality, we assume that the majority allele is labeled as 0 and the minority allele as 1. We recall that in the bi-allelic case, genotypes can be expressed as integer numbers between 0 and p, effectively counting the occurrences of minority alleles. Let G_i^s denote the genotype of sample s at variant v_i. If $G_i^{s'} = 1$ and $G_i^{s''} = 0$, we call v_i a *simplex-nulliplex* variant. Similarly, we call it a *simplex-simplex* variant if $G_i^{s'} = G_i^{s''} = 1$ and a *duplex-nulliplex* variant if $G_i^{s'} = 2$ and $G_i^{s''} = 0$. For each progeny sample s and variant v_i, let $D_s^i(0)$ and $D_s^i(1)$ be the number of occurrences of the major and minor allele among all reads of s, respectively. The genotypes and allele depths form the input of our phasing algorithm.

Figure 4.1 shows the pipeline of WHATSHAP POLYPHASE-GENETIC. We start by identifying variant types that are most informative for Mendelian inference rules – usually simplex-nulliplex variants because they contain a unique and easy-to-trace minority allele (see Section 4.2.1 for further reasoning). Each pair (v_i, v_j) of picked variants is scored by a Bayesian model, where we compute the support for co-locating the minority alleles on the same haplotype or for placing them on different ones. Every offspring sample s votes for either of two cases based on the observed allele depths $D_s^i(0)$, $D_s^i(1)$, $D_s^j(0)$, and $D_s^j(1)$ for the two variants. This results in a graph with one vertex per minority allele and a log-likelihood score as edge weights. Using the same cluster editing model as in Section 3.2, we obtain clusters of alleles that should be placed on the same haplotype for parent sample s'. This is very similar to the clustering stage in WHATSHAP POLYPHASE, but here we cluster alleles rather than reads. Like before, the resulting clustering does not necessarily match the ploidy. Thus, the final step is to assign clusters to haplotypes in a conflict-minimizing way, using an interval scheduling model (Section 4.2.3).

We emphasize that WHATSHAP POLYPHASE-GENETIC does not phase all kinds of variants, as it explicitly selects easy-to-phase variant types in the first step. This is a design choice, as the

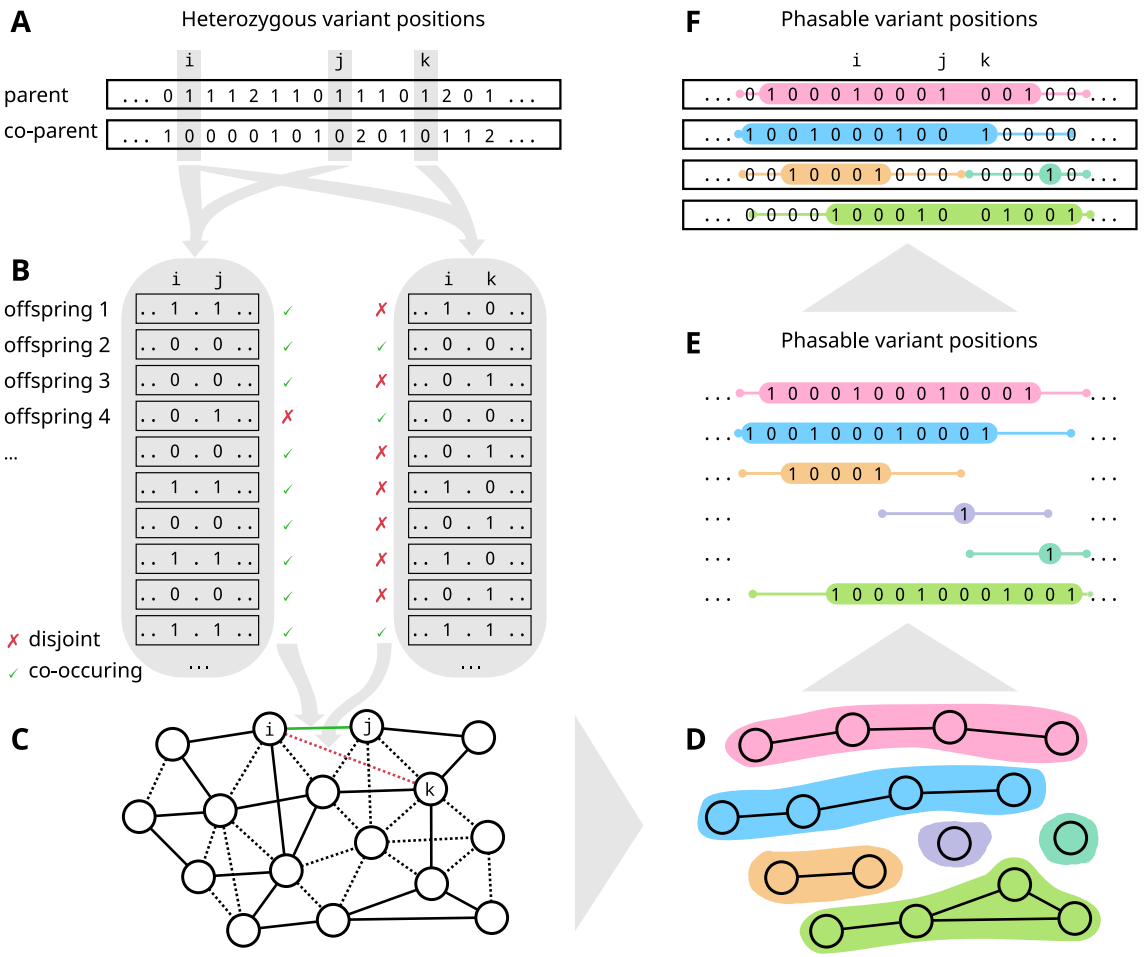
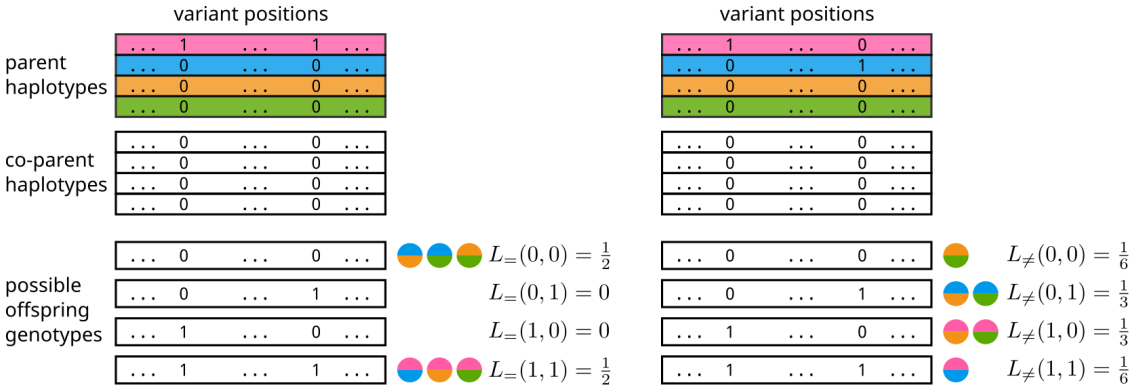


Figure 4.1: **Method overview.** **A**: Genotypes of both parent samples are scanned for informative variants. For illustration purposes, we focus on simplex-nulliplex variant pairs in this overview. **B**: Based on progeny allele depths (here we just show progeny genotypes for the sake of simplicity) every variant pair is either classified as having their minority alleles placed on the same haplotype (green check mark) or not (red cross). We compute a log-likelihood score for each considered pair of variants. **C**: We construct a graph with variants as nodes and insert a weighted edge for every previously computed score between two variants. **D**: A cluster editing model determines groups of variants whose minority alleles co-occur on the same haplotype. **E**: The clusters are embedded into the variant space to form (padded) intervals from the first to the last covered variant. **F**: We use an interval scheduling approach to select a maximum conflict-free subset that corresponds to p haplotypes (here $p = 4$).

omitted variants would introduce a lot of complexity into our model and only carry very weak signals. Earlier studies reported a relatively low fraction of multi-allelic variants among SNPs of less than 6% [34] and in our experiments, 40% of the bi-allelic variants turned out to be simplex-nulliplex. Our phasing method intends to compute a sparse phasing with gaps instead of resolving every variant position. The advantage of a genetic phasing method like WHATSHAP POLYPHASE-GENETIC is that it does not rely on reads providing sufficient connectivity between variants but instead utilizes the fact that offspring individuals share long coherent pieces of their DNA with their parents. The distance between variants, for which one can still find evidence on the co-occurrence of alleles on the same haplotype, is thus limited by the distance between recombination events and not by the lengths of the input reads.

In the evaluation of WHATSHAP POLYPHASE, we saw that the limited connectivity between variants eventually causes switch errors in the resulting haplotypes. Ideally, the herein pre-



(a) Markers on same haplotype (b) Markers on different haplotype

Figure 4.2: **Tetraploid heritage probabilities.** Example for $L_=$ and $L_\neq$ on tetraploid samples and two simplex-nulliplex variants. Each of the six possible haplotype pairs from the first parent leads to one of four possible genotype patterns (the other parent is homozygous). For parents, numbers indicate alleles. For offspring, numbers indicate genotypes, i.e. number of minority alleles.

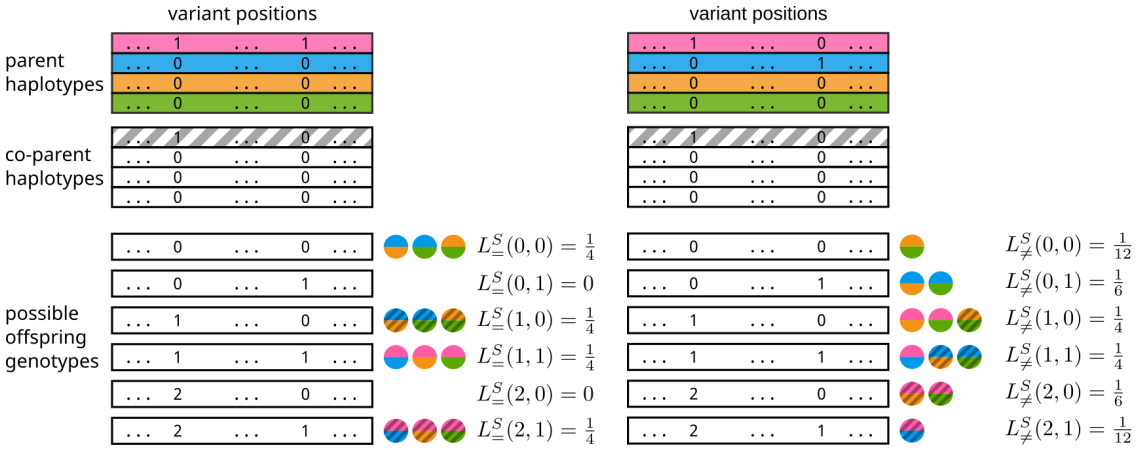
sented genotype-based phasing would be complemented with a read-based method to combine the best of both worlds: Long accurate phasing blocks and the ability to resolve all types of variants in between. This idea will be further discussed in Section 4.4.

4.2.1 Identifying and scoring phasable variants

Following the Mendelian rules, a progeny sample with even ploidy p inherits $\frac{p}{2}$ of its haplotypes from each of the two parents. Apart from recombination events, the two inherited haplotypes from one parent stay the same. This allows us to infer the co-occurrence of certain alleles on the parental haplotypes without directly incorporating sequencing information. Since we have to trace the origin of observed alleles among the progeny samples, only certain variant types can be phased with sufficient statistical evidence. We cover three different pairs of variant types that WHATSHAP POLYPHASE-GENETIC is able to process. All variant types not covered by any of these pairs remain unphased.

Two simplex-nulliplex variants

The easiest case is given by two simplex-nulliplex variants v_i and v_j, where s' has exactly one occurrence of the minority allele on any haplotype for each variant. We call these occurrences *markers* and denote the indices of the true haplotypes containing the markers (i.e. the minority alleles) with h_i and h_j, respectively. Every offspring sample either inherited both minority alleles, exactly one of them, or none of them with different probabilities, depending on whether $h_i = h_j$ or $h_i \neq h_j$. Let $L_=(n_i, n_j)$ and $L_\neq(n_i, n_j)$ be the probability for a progeny sample to inherit n_i and n_j minor alleles for variants v_i and v_j, given $h_i = h_j$ and $h_i \neq h_j$ respectively. If there are no recombination events between v_i and v_j, then $L_=$ and $L_\neq$ can be computed as



(a) Markers on same haplotype (b) Markers on different haplotype

Figure 4.3: **Tetraploid heritage probabilities.** Example for $L^S_=$ and $L^S_\neq$ on tetraploid samples. With the co-parent being heterozygous on the simplex-simplex variant, there are now six possible offspring genotype patterns. The striped circles indicate that the first haplotype from the co-parent was inherited.

follows:

$$L_=(1,1)=\frac{\binom{1}{1}\cdot\binom{p-1}{p/2-1}}{\binom{p}{p/2}}=\frac{1}{2} \qquad L_\neq(1,1)=\frac{\binom{2}{2}\cdot\binom{p-2}{p/2-2}}{\binom{p}{p/2}}=\frac{\frac{p}{2}-1}{2(p-1)}$$

$$L_=(0,1)=L_=(1,0)=0 \qquad L_\neq(0,1)=L_\neq(1,0)=\frac{\binom{2}{1}\cdot\binom{p-2}{p/2-1}}{2\cdot\binom{p}{p/2}}=\frac{p}{4(p-1)}$$

$$L_=(0,0)=\frac{\binom{1}{0}\cdot\binom{p-1}{p/2}}{\binom{p}{p/2}}=\frac{1}{2} \qquad L_\neq(0,0)=\frac{\binom{2}{0}\cdot\binom{p-2}{p/2}}{\binom{p}{p/2}}=\frac{\frac{p}{2}-1}{2(p-1)}$$

Figure 4.2 illustrates the probabilities $L_=$ and $L_\neq$ for two simplex-nulliplex variants in the tetraploid case, depending on whether $h_i = h_j$ or $h_i \neq h_j$.

Simplex-simplex and simplex-nulliplex variant

For the second case, we assume v_i to be a simplex-simplex variant, while v_j stays simplex-nulliplex. For v_i, each offspring has a 50%-chance to inherit a minority allele from s'' independently from what it inherited from s'. Therefore, it is now possible for an offspring to attain two minority alleles at v_i, even though both parents only possess one minority allele each. We modify the introduced functions $L_=$ and $L_\neq$ for the simplex-simplex case and call them $L^S_=$ and $L^S_\neq$. They are then defined as

$$L^S_=(q,r)=\frac{L_=(q,r)+L_=(q-1,r)}{2}$$
$$L^S_\neq(q,r)=\frac{L_\neq(q,r)+L_\neq(q-1,r)}{2},$$

where $q \in \{0,1,2\}, r \in \{0,1\}$. For consistency, we set $L_=(q,r) = L_\neq(q,r) = 0$ for $q \notin \{0,1\}$ because it is not possible to inherit two minority alleles from a simplex-nulliplex variant. The tetraploid example has been updated accordingly in Figure 4.3.

Comparing the two cases that the marker alleles for the parent sample reside on either the same or on different haplotypes, we can see that the likelihood margins for the same genotype patterns are strongly reduced compared to Figure 4.2. While before there was a margin of $\frac{1}{3}$ for all patterns, it is now at most $\frac{1}{6}$ and even 0 for two of the six genotype patterns.

Duplex-nulliplex and simplex-nulliplex variant

As a final case, we consider v_i to be a duplex-nulliplex variant, while v_j again remains a simplex-nulliplex variant. This combination is comparable in complexity to the second case, hence we implemented it for our model. For the sake of completeness, we state the corresponding likelihood functions $L^D_=$ and $L^D_{\neq}$ but do not continue the tetraploid example.

$$
\begin{aligned}
L^D_=(2,1) &= \frac{\binom{p-2}{p/2-2}}{\binom{p}{p/2}} = \frac{p/2-1}{2(p-1)} & L^D_{\neq}(2,1) &= 0 \\
L^D_=(2,0) &= 0 & L^D_{\neq}(2,0) &= \frac{\binom{2}{2}\cdot\binom{p-3}{p/2-2}}{\binom{p}{p/2}} = \frac{(p/2)(p/2-1))}{2(p-1)(p-2)} \\
L^D_=(1,1) &= \frac{\binom{1}{1}\cdot\binom{p-2}{p/2-1}}{\binom{p}{p/2}} = \frac{p}{4(p-1)} & L^D_{\neq}(1,1) &= \frac{\binom{2}{1}\cdot\binom{p-3}{p/2-2}}{\binom{p}{p/2}} = \frac{(p/2)(p/2-1))}{(p-1)(p-2)} \\
L^D_=(1,0) &= \frac{\binom{1}{1}\cdot\binom{p-2}{p/2-1}}{\binom{p}{p/2}} = \frac{p}{4(p-1)} & L^D_{\neq}(1,0) &= \frac{\binom{2}{1}\cdot\binom{p-3}{p/2-1}}{\binom{p}{p/2}} = \frac{(p/2)(p/2-1))}{(p-1)(p-2)} \\
L^D_=(0,1) &= 0 & L^D_{\neq}(0,1) &= \frac{\binom{1}{1}\cdot\binom{p-3}{p/2-1}}{\binom{p}{p/2}} = \frac{(p/2)(p/2-1))}{2(p-1)(p-2)} \\
L^D_=(0,0) &= \frac{\binom{p-2}{p/2}}{\binom{p}{p/2}} = \frac{p/2-1}{2(p-1)} & L^D_{\neq}(0,0) &= \frac{\binom{p-3}{p/2}}{\binom{p}{p/2}} = \frac{(p/2-1)(p/2-2)}{2(p-1)(p-2)}
\end{aligned}
$$

We argue that including more complex combinations of variant types would add very little additional signal to our computation, as the likelihood margins for the resulting genotype patterns diminish further and further. We refer to all variants of type simplex-nulliplex, simplex-simplex, or duplex-nulliplex as *phasable variants* and to all pairs of phasable variants that match one of the three presented cases as *phasable variant pair*. Note that two variables can be phasable but not be considered a phasable pair, e.g. two simplex-simplex variants.

Assigning scores

For each phasable pair of variants v_i, v_j we want to compute the likelihood for two hypotheses: (i) $h_i = h_j$, i.e., the marker alleles for v_i and v_j reside on the same haplotype, and (ii) $h_i \neq h_j$, i.e., the marker alleles do not share haplotypes. Duplex-nulliplex variables are a special case, because we get two marker alleles for a single variant, say v_i. In that case, the hypothesis $h_i = h_j$ is interpreted as the marker allele for v_j to co-occur on the same haplotype as any of the two marker alleles for v_i (or co-occur with none of them for $h_i \neq h_j$).

To achieve this, we determine how well the observed allele depth D^i_s and D^j_s for each offspring sample s is explained by either of the two hypotheses. The low coverage of the offspring samples yields inaccurate genotype estimations which incentivizes us to rather work

on the allele depths directly. We can see the necessary computation for the first hypothesis and a single offspring s in Equation (4.1). For each possible genotype pair g_i, g_j for s on variants v_i, v_j, we take the likelihood of observing the allele depths of s, assuming that g_i and g_j are the true genotypes. Because not all genotype pairs are equally likely, we use $L_=(g_i, g_j)$ as prior probabilities for each genotype pair (or $L^S_=(g_i, g_j)$ or $L^D_=(g_i, g_j)$, depending on the variant type of v_i). The likelihood of observing certain allele depths only depends on the genotype for the respective variant position. Therefore, we can split the first factor in each summand into a product covering both variants v_i and v_j independently.

$$\begin{aligned} P(D^i_s, D^j_s \mid h_i = h_j) &= \sum_{g_i, g_j \in \{0,\dots,p\}} P(D^i_s, D^j_s \mid G^s_i = g_i, G^s_j = g_j) \cdot L_=(g_i, g_j) \\ &= \sum_{g_i, g_j \in \{0,\dots,p\}} P(D^i_s \mid G^s_i = g_i) \cdot P(D^j_s \mid G^s_j = g_i) \cdot L_=(g_i, g_j) \end{aligned} \tag{4.1}$$

The likelihood to observe D^i_s given a genotype $g_i \in \{0, \dots, p\}$ follows a binomial distribution as shown in Equation (4.2) where $B_{\text{pmf}}(n, k, p)$ denotes the binomial probability mass function:

$$P(D^i_s \mid G^s_i = g_i) = B_{\text{pmf}}\left(\underbrace{D^i_s(0) + D^i_s(1)}_{\text{coverage}}, \underbrace{D^i_s(1)}_{\text{number of min. alleles}}, \underbrace{\frac{g_i}{p}}_{\text{prob. of min. allele}} \right) \tag{4.2}$$

The analogous case for $h_i \neq h_j$ uses $L_\neq$ instead of $L_=$, but follows the same scheme.

Influence of recombination

When introducing the functions $L_=$ and $L_\neq$, we ignored the presence of recombination events. A single recombination event causes one of the haplotypes of an offspring sample s to inherit from different parental haplotypes on either side of the event location. Thus, the proposed probabilities for $L_=$ and $L_\neq$ (and likewise for the other functions) only hold under the assumption that no recombination event occurred between the two considered variants. However, without knowledge about recombination rates (that might even differ locally), it is not possible to correct the probabilities accordingly. Usually, recombination events are quite rare with less than 10 events per chromosome. If we apply $L_=$ and $L_\neq$ only to variants with moderate distance on the genome, the results have a high probability of being correct. We will keep this issue in mind and discuss it after introducing the clustering stage of the algorithm.

4.2.2 Clustering variants based on Bayesian scores

After all phasable variant pairs are known, the next step is to create an allele graph that contains one node for each marker allele. For every phasable variant pair v_i, v_j, we connect all

marker alleles for v_i with all marker alleles for v_j by a weighted edge. For duplex-nulliplex variants, we insert an edge with weight $-\infty$ for the two corresponding marker alleles. All other pairs of nodes are connected with a zero-weighted edge. We recall here that this method only phases one of the parents. Thus, we only have a single node for simplex-simplex variants, while duplex-nulliplex variants induce two nodes.

The idea is very similar to the read clustering in Section 3.2: We want to identify clusters of marker alleles that are likely to co-occur on the same haplotype. As before, we use the cluster editing model to transform the graph into a clique graph over the non-negative edges. The result is a clustering with possibly more than p clusters, which we process in the next stage of the algorithm. Duplex-nulliplex variants contain two indistinguishable marker alleles that need to be assigned to one cluster each. For that reason, we insert one node for each marker allele instead of each variant; this allows a duplex-nulliplex variant to be associated with two haplotypes while keeping our graph construction compatible with cluster editing. Two alleles from the same variant can obviously not reside on the same haplotype, hence the forbidden edges between all node pairs of this kind.

Scoring scheme

In order to complete the graph modeling, we now define how the edge weights are computed to provide the intended property of scoring co-occurrence likelihoods between allele pairs. At the top level, the weight $s(i,j)$ of an edge between nodes i and j can be expressed as a logarithm over the ratio of two likelihoods. We already presented this technique for the read scoring scheme in Section 3.2.2. The two likelihoods refer to the hypotheses that (i) the marker alleles of a variant v_i share any haplotypes with the marker alleles of variant v_j (denoted as $h_i = h_j$) and (ii) that the two sets of marker alleles are disjoint regarding haplotype assignment (denoted as $h_i \neq h_j$); both depend on the observed allele depth among the offspring. If v_i has two marker alleles (and thus is associated with two nodes i_1 and i_2), both nodes will have the same neighborhood, i.e., $s(i_1,j) = s(i_2,j)$ for all other nodes j.

$$
\begin{aligned}
s(i,j) &:= \log\left(\frac{P(h_i = h_j \mid D^i_{s_1},\ldots,D^i_{s_p},D^j_{s_1},\ldots,D^j_{s_p})}{P(h_i \neq h_j \mid D^i_{s_1},\ldots,D^i_{s_p},D^j_{s_1},\ldots,D^j_{s_p})}\right) \\
&= \log\left(\frac{\frac{P(D^i_{s_1},\ldots,D^i_{s_p},D^j_{s_1},\ldots,D^j_{s_p}\mid h_i=h_j)\cdot P(h_i=h_j)}{P(D^i_{s_1},\ldots,D^i_{s_p},D^j_{s_1},\ldots,D^j_{s_p})}}{\frac{P(D^i_{s_1},\ldots,D^i_{s_p},D^j_{s_1},\ldots,D^j_{s_p}\mid h_i\neq h_j)\cdot P(h_i\neq h_j)}{P(D^i_{s_1},\ldots,D^i_{s_p},D^j_{s_1},\ldots,D^j_{s_p})}}\right) \\
&= P(D^i_{s_1},\ldots,D^i_{s_p},D^j_{s_1},\ldots,D^j_{s_p} \mid h_i = h_j) \\
&\quad - P(D^i_{s_1},\ldots,D^i_{s_p},D^j_{s_1},\ldots,D^j_{s_p} \mid h_i \neq h_j) + \log\left(\frac{1}{p-1}\right)
\end{aligned}
\tag{4.3}
$$

The full scoring equation is given in Equation (4.3). Each hypothesis is expressed as a conditional probability with the observed allele depths over all offspring samples as conditions.

We apply Bayes' theorem to express the probabilities as likelihoods of allele depths depending on either $h_i = h_j$ or $h_i \neq h_j$ and resolve both the fraction and the logarithm. The prior probabilities $P(h_i = h_j)$ and $P(h_i \neq h_j)$ are $\frac{1}{p}$ and $\frac{p-1}{p}$, respectively.

Since the allele depths of different offspring samples are independent of each other, we can decompose the conditioned probabilities into products of probabilities over a single offspring sample each (see Equations (4.4) and (4.5)). These factors are identical to the probabilities we already computed in Equation (4.1).

$$P(D^i_{s_1}, \ldots, D^i_{s_p}, D^j_{s_1}, \ldots, D^j_{s_p} \mid h_i = h_j) = \prod_{l=1}^{p} P(D^i_{s_l}, D^j_{s_l} \mid h_i = h_j) \tag{4.4}$$

$$P(D^i_{s_1}, \ldots, D^i_{s_p}, D^j_{s_1}, \ldots, D^j_{s_p} \mid h_i \neq h_j) = \prod_{l=1}^{p} P(D^i_{s_l}, D^j_{s_l} \mid h_i \neq h_j) \tag{4.5}$$

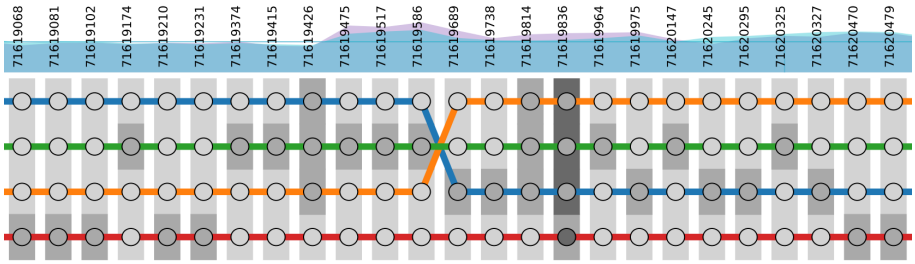
This concludes the construction of the allele graph. For solving techniques of the underlying cluster editing model, we refer to the explanations in Section 3.2.

Windowed scoring

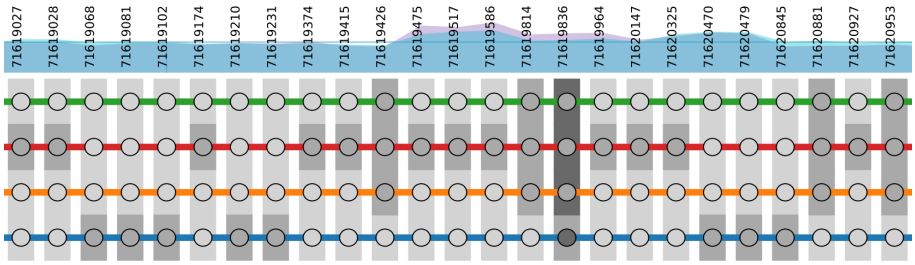
A full pair-wise scoring between all phasable variants requires a quadratically growing number of computations, rendering this process intractable for chromosome-scale phasing. We therefore use a *scoring window* W as the maximal distance between two variants, for which we compute a score between the associated marker alleles. The distance is counted in intermediate phasable variants, i.e. the marker alleles of two variants will only be scored if at most $W - 1$ other phasable variants are in between. Otherwise, we apply a neutral score of 0.

The downside of windowed scoring is the loss of connectivity information for distant variant pairs, which increases the risk of switch errors due to locally (but not globally) optimal clustering. To keep the window size as large as possible, we introduce a compromise between maximum variant distance and computational efficiency: Instead of computing a score for all neighboring variants inside the window, we use a sparse scoring pattern. On average, our pattern scores every 6th possible variant pair: For every phasable variant v_i, we consider the $\lceil \frac{W}{24} \rceil$ next and previous phasable variants (regarding their ordered positions on the genome). From there on, we select every third variant until another $\lceil \frac{W}{24} \rceil$ phasable variants are taken on each side. We proceed with every seventh variant for the next$\lceil \frac{W}{24} \rceil$ variants on each side and then select every 13th variant until the bounds of the window are reached. This allows us to increase W six-fold with the same number of scoring computations.

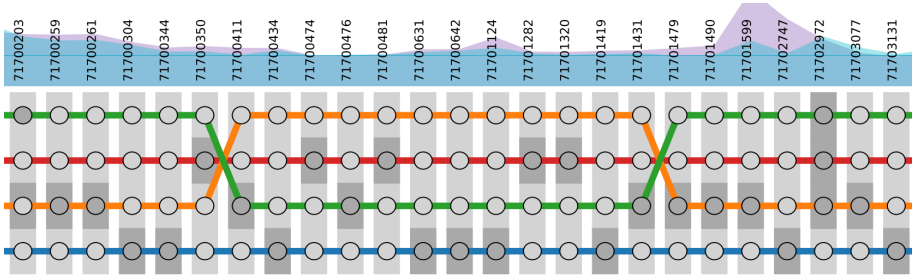
The sparse pattern consistently improved the phasing quality during our tests. We conclude that connecting a few distant variants via scoring adds more valuable information to the graph than fully connecting all variants in a smaller window. The effect of a spread-out scoring window is visualized in Figures 4.4a and 4.4b with the former displaying a phased region computed with a filled, short scoring window and the latter the same region but computed with a sparse window. The spatially more confined window results in a switch error (e.g. due to



(a) Switch error inside the genetic phasing.



(b) Same region without switch error.



(c) Combination of two switch errors canceling each other.

Figure 4.4: **Switch errors in genetic phasing.** This figure shows example regions from three different phasings. Each variant is drawn as a column, stating its genome position and its four ground-truth alleles as four rectangles in different shades of gray. The four colored lines represent the predicted haplotypes with the circles on the lines being the alleles of the respective haplotypes. When the tone of a circle matches the tone of the rectangle behind it, the predicted haplotype matches the ground truth, other the phasing contains a (flip) error. When colored lines switch their rows, this means that the cheapest switch-flip transformation (i.e. the transformation that is computed to determine the SFR) contains switch errors between the respective variant positions. **(a)** The predicted phasing was computed with a consecutive scoring window of size 250. When computing the SFR between predicted and true phasing, the blue and orange haplotypes switch positions, indicating a switch error in the phasing. **(b)** Here, a sparse scoring window of size 1500 was used with the same number of scoring computations as the previous example. The switch error was avoided through the longer scoring range. Note that the haplotype order might be different and that the variant positions are not congruent, because the differences in scoring resulted in a different set of phased variants in the results. **(c)** This again shows the phasing with a sparse window but at a different location, on which two switch errors cancel each other out.

collapsed regions), while the sparse window prevents this error, despite using the same number of scoring partners for each variant. However, even large score windows do not prevent all switch errors, as shown in Figure 4.4c. We can see that the phasing contains two switch errors that cancel each other out. Even though introducing one of the switches might locally be the most likely solution to the phaser, it recovers from this mistake through long-range scores in its allele graph. As default, we choose $W = 1500$ with 250 scoring partners for each variant in both directions, because it proved to be a good compromise between speed and accuracy in practice.

Influence of recombination

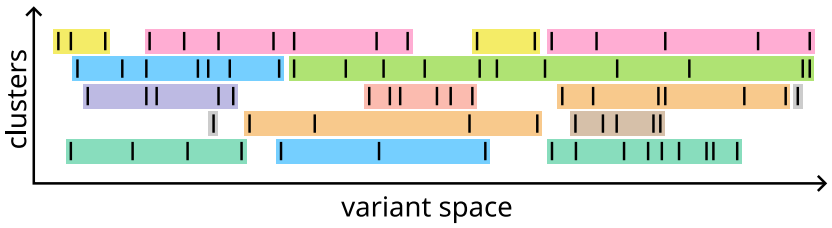
We still have to discuss the influence of recombination events on our scoring scheme. Considering that the average distance between two consecutive simplex-nulliplex is in the dimension of 100bp, the distances between scored variants are around hundreds of kb for the default window size. The exact recombination rate of potato cultivars highly depends on the exact breed, the chromosome, and the position therein but a rough estimate is that it does not exceed a rate of 2cM/Mb (centimorgan per million bases) in most cases [107, 108]. This is equivalent to about one recombination every 50Mb, which is more than 100 times larger than the average distance between scored variants. Thus, when aggregating the score of one variant pair over all offspring samples, we can expect about 1% of them to yield a false calculation due to a recombination event between the two variants. We argue that the influence of these recombination events is low enough for our model to function in the intended way. It would be beneficial to incorporate information about local recombination rates, but since such information was not available to use by the time we developed WHATSHAP POLYPHASE-GENETIC, we left this issue open for further research.

4.2.3 Assigning haplotypes: Interval scheduling

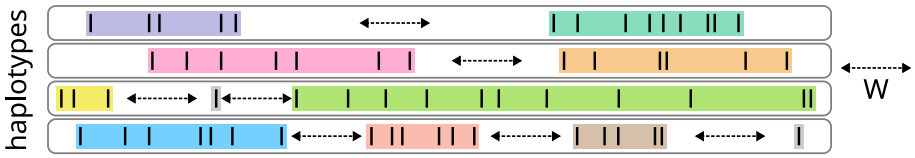
Cluster editing does not necessarily yield exactly p clusters, which would directly result in a phasing of all phasable variant pairs. In practice, the number of clusters is much higher with many small and even singleton clusters due to different sorts of errors and noise in the data. There are two ways to deal with this issue: We could either find an assignment for all clusters to the p haplotypes, such that the contradiction to the scores is minimized or we could find a maximum conflict-free subset of clusters which explains the highest possible number of variants. To be consistent with the previous design choices of specifically selecting variant types that fit our statistical model, we decided to implement the second solution.

Let $\mathcal{C} := \{C_1, \dots C_q\}$ be the set of computed clusters from the previous step and let $\min(C_i)$ and $\max(C_i)$ be the lowest and highest index for all variant indices in C_i for $1 \leq i \leq q$, respectively. If two clusters C_i, C_j do not overlap and there is at least a full scoring window W of phasable variants between them, i.e. either $\max(C_i)+W \leq \min(C_j)$ or $\max(C_j)+W \leq \min(C_i)$ holds, these clusters are compatible and can be assigned to the same haplotype. If C_i and C_j do not overlap but the gap between them is smaller than W phasable variants, then variants from both clusters have been scored against each other but the clustering model still determined a split into two clusters as the best solution. In this case, we assume that the variants from C_i and C_j have to be assigned to different haplotypes.

As an illustration, Figure 4.5 shows an example of four haplotypes. The clusters at the top have been computed by the clustering stage and contain sets of marker alleles with a certain (horizontal) position inside the variant space. For the haplotype assignment, a subset of clusters is selected such that there is no overlap and at least a distance of W between all clusters on the same haplotype. The goal is to find an assignment of each cluster to either



(a) Allele clusters embedded into the variant space



(b) Clusters assigned to haplotypes with at least distance W to one another

Figure 4.5: **Example for cluster scheduling. (a)** Clusters are drawn as colored bars with contained marker alleles as black stripes. The horizontal position indicates the position of each allele in the variant space. **(b)** For a ploidy of 4, a subset of clusters is assigned to the four haplotypes. Clusters must not overlap and leave a horizontal space of at least W between them. The selection maximizes the number of contained marker alleles.

one of the k haplotypes or to remain unphased, such that the total number of assigned marker alleles is maximized. For instance, the orange cluster with four marker alleles in the fourth row is not placed on the first haplotype because it is too close to the purple and turquoise clusters. The turquoise cluster on the first haplotype could be replaced by the magenta cluster on the right side which has a larger horizontal span. However, since we optimize for the number of assigned marker alleles, this replacement would result in an overall worse solution.

In scheduling theory, the described optimization problem is known as weighted interval scheduling on p identical machines. Each cluster C_i corresponds to a job with fixed start time $\min(C_i)$, fixed end time $\max(C_i)$, and profit w_i, representing the number of contained marker alleles. Arkin and Silverberg developed both a formulation as an Integer Linear Program (ILP) and as a minimum cost flow [109]. They also point out that the matrix of the constraint coefficients is unimodular, such that the ILP is solvable in polynomial time.

Here, we use an alternative and easy-to-implement ILP formulation, which can still be solved efficiently in practice. It contains a set of binary variables x_i^j for $1 \leq i \leq q, 1 \leq j \leq p$, where q is the number of clusters. If cluster i is assigned to haplotype j, x_i^j is set to 1 and 0 otherwise. Let $X := \{(i,l) \mid C_i \text{ incompatible to } C_l\}$. Then an optimal cluster assignment is found by solving the following ILP:

$$\max \quad \sum_{i=1}^{q}\sum_{j=1}^{p} x_i^j w_i \tag{4.6}$$

$$\text{subject to} \quad x_i^j + x_l^j \leq 1 \qquad \forall\, 1 \leq j \leq p, \forall\, (i,l) \in X \tag{4.7}$$

$$\sum_{j=1}^{p} x_i^j \leq 1 \qquad \forall\, 1 \leq i \leq q \tag{4.8}$$

$$x_i^j \in \{0,1\} \qquad \forall\, 1 \leq i \leq q, \forall\, 1 \leq j \leq p \tag{4.9}$$

After the optimal haplotype assignment is computed, we report all assigned alleles as phased in the output VCF file. Our method does not offer any routine to determine cut positions because we assume the entire chromosome to be connected through heredity information.

4.3 Experiments

The structure of this section and some of the phrasing are taken from [16].

We conducted our benchmarks on two parent samples of *Solanum tuberosum*, named "Altus" and "Colomba", and 193 offspring samples. Each of the samples has been sequenced using Illumina sequencing technology with 250bp paired-end reads. The average sequencing depth is ∼6× for each offspring sample and more than 300× for each parental sample. We aligned all reads to the Solyntus V1.1 reference genome [50] and performed a variant calling using GATK [37]. In addition, we have a library of HiFi reads for Altus with an average coverage of 24× per haplotype.

In order to evaluate the accuracy of our method, we used the HiFi reads to create four ground truth haplotypes for small stretches of the genome. We computed an assembly graph over these reads using hifiasm v0.13 [56] with standard settings, aligned the node sequences to the reference genome, and selected three regions on chromosomes 3, 4, and 5 that were continuously covered by four contigs each. Despite their relatively small size of about 300kb, they were among the largest of their kind, as it proved quite difficult to find long regions with four clearly visible haplotypes based on the assembly alone. We extracted these regions from the VCF files and both the HiFi and Illumina BAM files. In the following, we will refer to these regions by just stating the chromosome number, i.e., chromosome 3, 4 or 5.

To our knowledge, there is no other method that utilizes the same type of input as WHATSHAP POLYPHASE-GENETIC. The variety of existing phasing algorithms we mentioned in Chapter 3 is designed to compute phasings from reads of a single individual. The genetic polyploid algorithms TriPoly [42] and PopPoly [43] are closest to our method in terms of input data but still require input reads for all offspring samples, and are thus not directly applicable to our data.

We used the same machine and workflows to execute the experiments as in the previous chapters.

4.3.1 Evaluation on HiFi-assembled regions

I repeated the experiments from [16] for the full population runs and newly created plots for this. I added one setting without retyping and an explanation of what the retyping does.

We ran WHATSHAP POLYPHASE-GENETIC on all three regions with the default scoring window of 250 (or 1500 if one counts the gaps as well) and three different variant selections: (i) only simplex-nulliplex variants, (ii) additional simplex-simplex variants and (iii) additional

duplex-nulliplex variants, i.e., all three implemented types. We refer to these selections by their indices 1, 2, and 3. We only included SNPs in our experiments because they can be most reliably called from short reads.

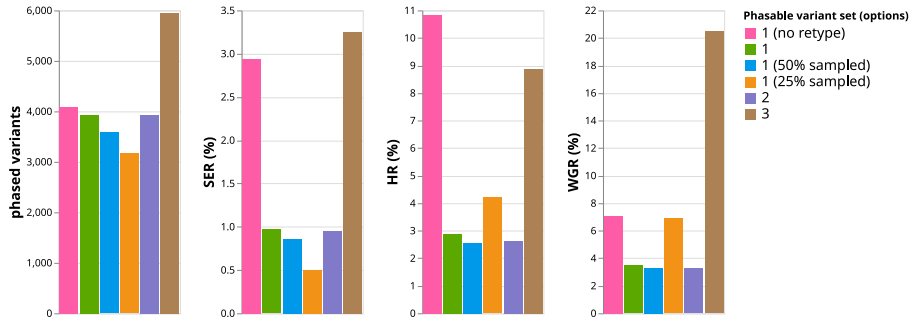
Like the read-based version, WHATSHAP POLYPHASE-GENETIC has an optional flag that allows overriding the genotypes provided by the VCF but does not consider any variants marked as homozygous in the VCF. Since the genotypes impact the selection of phasable variants, the optional flag also triggers a retyping of genotypes before variants are classified for the actual phasing. The retyping selects the most likely genotype configuration for the parents regarding how well the configuration explains the observed allele depths among the offspring samples. Only the variants that are retyped into a suitable genotype configuration (depending on whether variant type subsets 1, 2, or 3 were chosen) will be kept for further computations. All other variants remain unchanged in the VCF.

We used the `--distrust-genotypes` option for most of our experiments, as it improved the results for all three test regions. We included one set of experiments with the flag disabled to show the impact of the retyping. As error metrics, we use the Hamming rate (HR), switch error rate (SER), and wrong genotype rate (WGR) that we defined in Section 1.5. Additionally, we also report the number of phased variants.

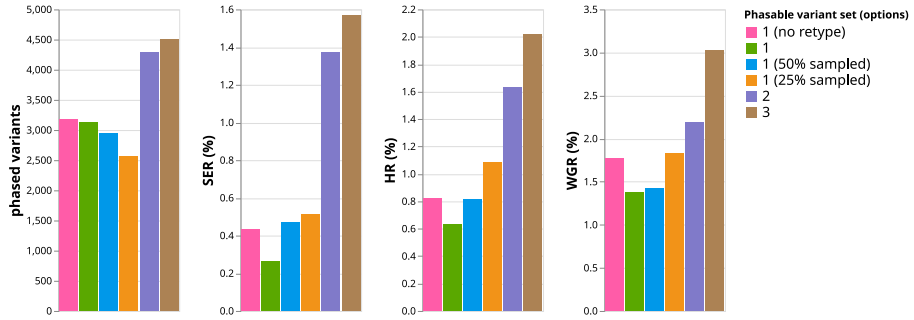
The results for all three regions are summarized in Figure 4.6. The sets of phasable variant types are indicated by numbers 1, 2, and 3, as explained above. As expected, including more variant types increases the number of phased variants. However, the more complex variant types also result in higher error rates. Especially the inclusion of duplex-nulliplex variants (on top of simplex-simplex variants) causes a three-fold increase in the HR on all three regions. This can be mostly explained by the even larger increase in wrong genotypes among the phased variants. In comparison, the addition of only simplex-simplex variants shows a moderate but still noticeable increase in error rates. On chromosomes 3 and 5, the error rates are very close to the setting with simplex-nulliplex variants only, but the increase in additionally phased is also only minimal. On chromosome 4, there are a lot more simplex-simplex variants, but therefore all error metrics degrade substantially with this variant type included.

In general, the Hamming rates can be considered quite low for simplex-nulliplex variants with less than 1% on chromosomes 4 and 5. This proves the overall correctness of the computed phasing. For chromosome 3, the HR grows to almost 3% which can be explained by the elevated genotype divergence compared to the other two regions. Depending on the chromosome, WHATSHAP POLYPHASE-GENETIC is able to phase about 25-40% of the biallelic variants with these settings. The variant retyping improves all tested metrics substantially for simplex-nulliplex variants except for the HR for chromosome 4 and a slight loss of phased variants in chromosomes 3 and 4. Therefore, we enabled the retyping for all other conducted tests. The exact numbers underlying the plots are provided in Supplementary Tables C.1, C.2, and C.3.

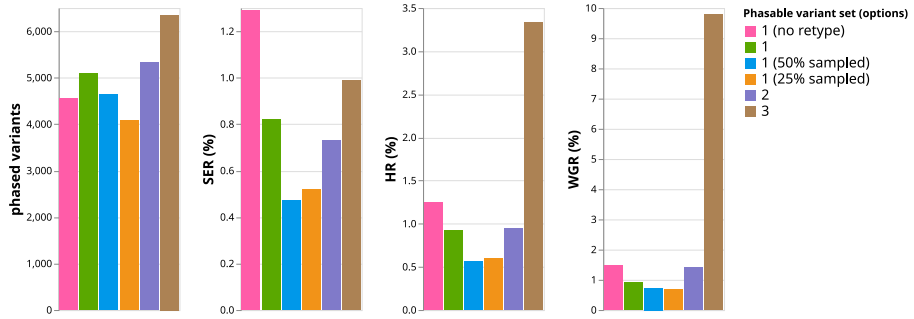
As the parental coverage was relatively high in the initial runs, we repeated the experiments for the simplex-nulliplex instances, but only used 50% and 25% of the parental reads for genotyping, respectively. That is, we used GATK to downsample the parental read data to



(a) Results for chromosome 3 (region ch03:60,269,000-60,504,000) with 9549 bi-allelic and 10286 total variants.



(b) Results for chromosome 4 (region ch04:71,586,000-71,947,000) with 12378 bi-allelic and 14500 total variants.



(c) Results for chromosome 5 (region ch05:56,711,000-57,066,000) with 13030 bi-allelic and 15810 total variants.

Figure 4.6: **Evaluation of WHATSHAP POLYPHASE-GENETIC.** The algorithm was run on three selected regions using various different configurations. The numbers indicate the phasable variant set (1: only simplex-nulliplex variants, 2: additionally simplex-simplex variants, 3: all three implemented types). The first set was tested without retyped variants and with genotype calls from only 50% and 25% of the Illumina reads, respectively.

50% and 25% on chromosomes 3, 4, and 5 and reran WHATSHAP POLYPHASE-GENETIC on the newly called variants. The number of phased variants decreases consistently with lower coverage, resulting in a total decline of about 20%. The main cause are genotype shifts during the variant calling due to different (and less) read information. Simplex-nulliplex variants in the full data set shift into another variant much more often than the other way around. The error rates follow no clear pattern throughout the coverage reduction. One would expect them to grow along with the uncertainty of the variant call like for the chromosome 4 region, but the other two regions rather see lower error rates with lower coverage. Aside from observing this

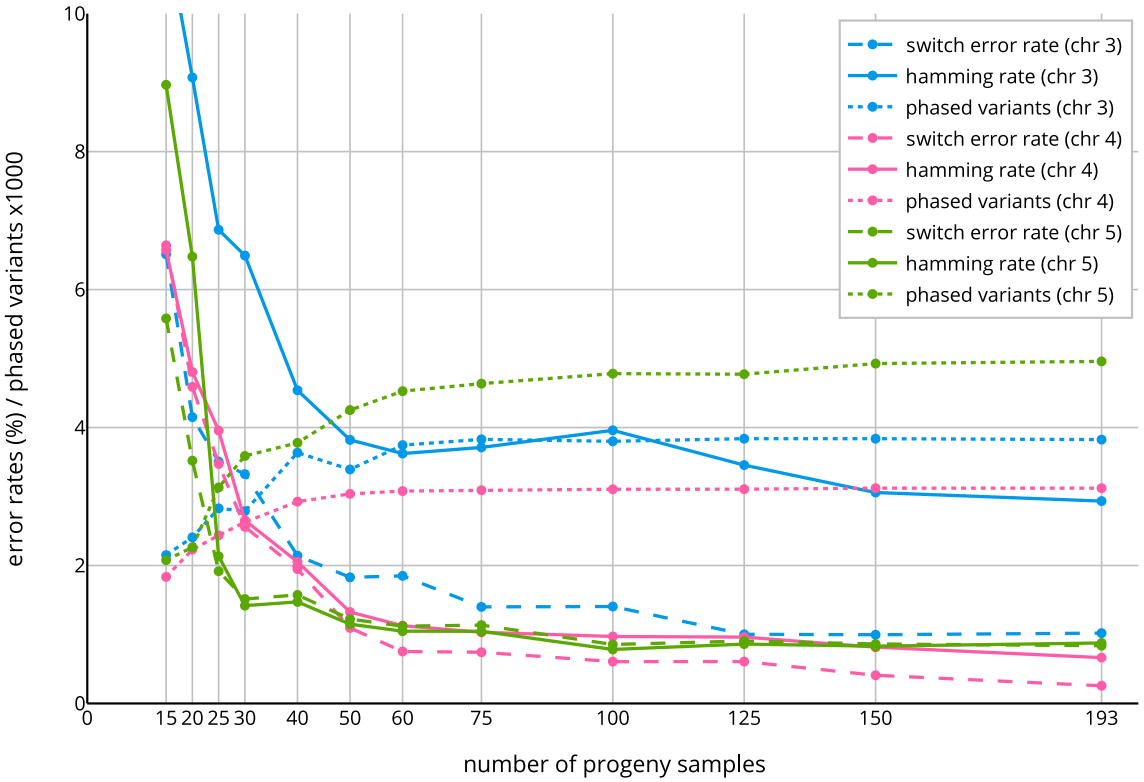


Figure 4.7: **Degradation of phasing accuracy with smaller offspring pool.** Shows SER, HR, and number of phased variants (y-axis) for different offspring pool sizes (x-axis) on the three validation regions (colors). Each point represents the mean value of the 10 random samples for each of the sample sizes. Error rates are shown as percentages, variant counts as thousands.

phenomenon by chance due to a single downsample experiment instead of multiple ones, one explanation (which we could neither prove nor reject) could be that the remaining simplex-nulliplex variants are more stable and easier to phase.

We further explored how dependent the phaser is on the number of offspring samples and parental sequencing depth. From the 193 offspring samples, we drew 10 random subsamples of sizes between 15 and 150 and reran the experiments for simplex-nulliplex variants only. The results for the three regions and three selected metrics are summarized in Figure 4.7. As expected, all error rates increase with smaller samples. Especially samples with less than 60 offspring begin to fall off from the rest.

4.3.2 WH-PPG scales to whole chromosomes

We ran WHATSHAP POLYPHASE-GENETIC on all twelve chromosomes of *Solanum tuberosum* to show its scalability to full genomes. We report the runtime and memory consumption since there exists no haplotype-resolved assembly of our sample to which we can compare the resulting phasing.

Figure 4.8 shows some statistics about the whole-chromosome runs. There is, on average, one simplex-nulliplex variant every 100bp, of which WHATSHAP POLYPHASE-GENETIC phased about 80%. All chromosomes were phased separately in parallel, running on a single core each. This results in a total of 300 CPU hours for the entire genome where chromosome 3

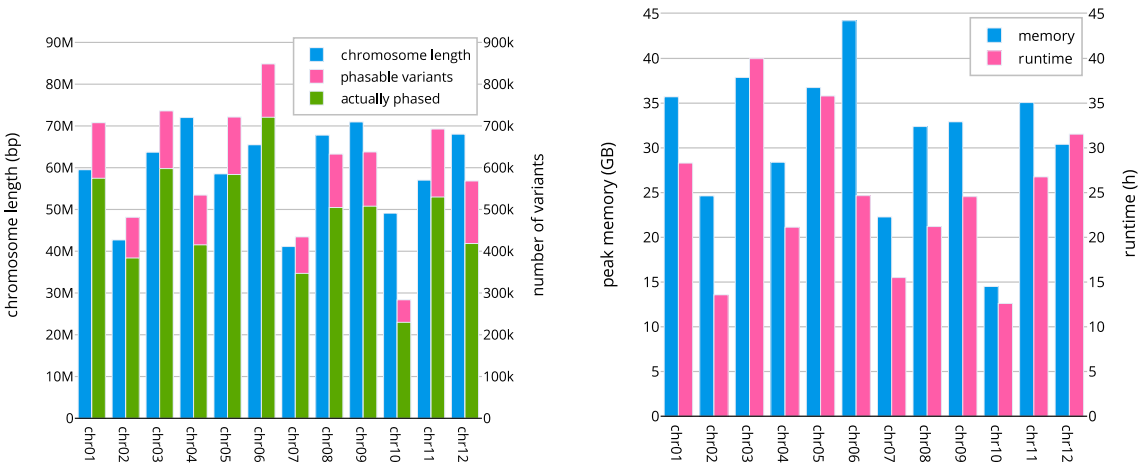


Figure 4.8: **Results on whole chromosomes.** Left: Length and number of simplex-nulliplex variants per chromosome with smaller green bars indicate the fraction of actually phased variants. Right: Used resources per chromosome.

had the highest running time (40 hours) and chromosome 6 had the highest peak memory consumption (44GB).

A strength of genetic phasing is the reduced dependency on read connectivity to produce long haplotype blocks. To evaluate this, we mapped all HiFi reads against the reference genome to get an estimate of what read-based phasers could achieve with the currently available data. We removed all intervals of size at least 10kb and with coverage less than $k = 4$ from the reference genome, as the HiFi reads will likely not be able to connect four haplotype blocks on each side of these intervals but instead induce at least one cut position each. If we order the remaining connected components by size, we can estimate the minimal block size of a best-case phasing that covers a fraction x of the phased chromosome for $0 \leq x \leq 1$. Note that the block size is equivalent to the N50 block size for $x = 0.5$. These estimates are summarized for each chromosome in Figure 4.9. Covering the relative size of the largest block is always possible with the largest block itself, hence the starting point (x_i, x_i) for each chromosome (with x_i being the relative size of the largest block on chromosome i). Except for chromosomes 7 and 10, the N50 block size would be below 20% of the chromosome size and below 5% if 90% of the chromosome are to be covered.

4.4 Integrating genetic and read-based phasing

This section has been newly added. The discussed topic has been mentioned as future work in [16] without any further details.

In Chapter 3 we evaluated WHATSHAP POLYPHASE and realized that many computed phasings contained critical switch errors. On the contrary, it is not restricted in what types of variants it can phase. These complementary strengths and weaknesses of WHATSHAP POLYPHASE-

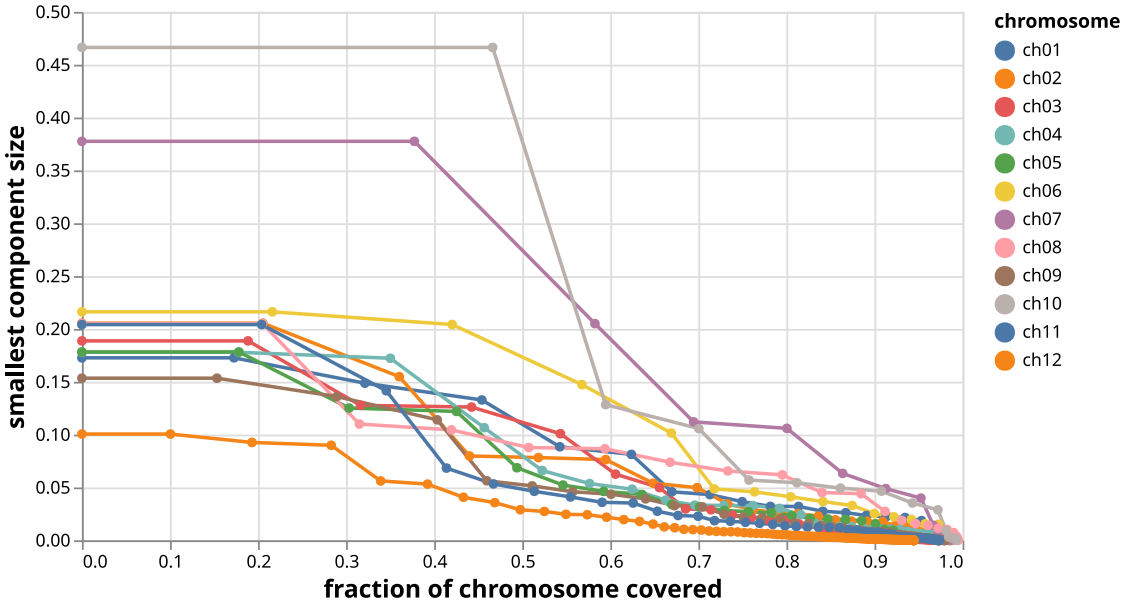


Figure 4.9: **Block size estimates for different chromosome coverages.** The x-axis represents the fraction of the chromosome covered by blocks and the y-axis shows the minimum block size (relative to the full chromosome) needed to reach the corresponding chromosome coverage.

GENETIC raised the question of whether both methods can be combined if both genotype data from offspring and sequencing data for the parents are available.

Both methods use fundamentally different models: WHATSHAP POLYPHASE clusters reads based on their allele similarity and subsequently works with the clusters to compute a phasing from left to right (with respect to genome positions). On the other side, WHATSHAP POLYPHASE-GENETIC clusters alleles and solves a global assignment problem to form haplotypes. Therefore, it is not obvious how to combine both methods into a single model. We propose two ideas on how the two methods can interact, one of which is implemented in the tool itself. Both ideas require that WHATSHAP POLYPHASE-GENETIC is run first and its output directly used in WHATSHAP POLYPHASE.

Sparse haplotypes as pre-phasing

The first idea is to use the sparse phasing from WHATSHAP POLYPHASE-GENETIC as pre-phasing. We defined this concept in Section 1.4 and described how it can be leveraged for reordering phase blocks in Section 3.4.3. In short, we run the purely read-based WHATSHAP POLYPHASE algorithm until we obtain phase blocks with ambiguous connections to their neighbors through collapsed regions or unconnected variants. For each haplotype block within, we compute a similarity to each of the sparse haplotypes based on common variants to find an optimal global assignment of haplotype blocks to one of the sparse haplotypes.

To test this hybrid procedure of read-based and genetic phasing, we used the high-quality regions from Section 4.3 and the respective pre-phasings obtained by WHATSHAP POLYPHASE-GENETIC for (i) simplex-nulliplex variants only and for (ii) all three implemented variant types. The pre-phasings were passed to WHATSHAP POLYPHASE as VCF input and the flag

`--use-prephasing` was enabled. Based on our observations from Section 3.5.3, we also added the flag `--distrust-genotypes` to account for the large genotype divergence between the variant calling from Illumina reads and the assembly-based ground-truth phasing from HiFi reads. As read input, we both used the HiFi reads (described in Section 3.5.3) for the respective regions, as well as the Illumina reads originally used for the variant calling (described at the beginning of Section 4.3). To avoid any intermediate cut positions introduced by disconnected variants, we used the lowest block cut sensitivity `-B 0` on all hybrid runs. As a baseline, we ran WHATSHAP POLYPHASE without pre-phasings (called "read-based") once with the same block cut sensitivity `-B 0` and once with `-B 1` to show how the disconnected variants affect the overall phasing contiguity.

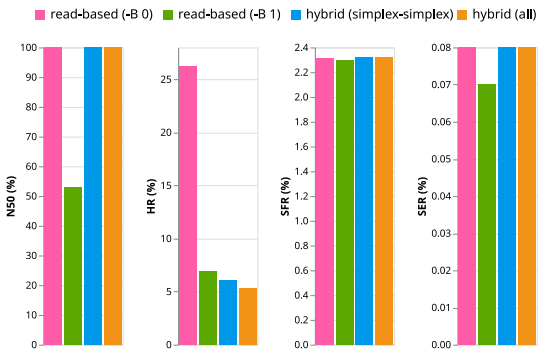
In Figure 4.10, we see the results for all three tested regions and two read datasets each. The differences between read-based and hybrid phasing are rather small in general. The SER and SFR are on par for the same block-cut policy over all datasets. Since the first two stages of WHATSHAP POLYPHASE are identical in both read-based and hybrid phasing, the additional information is only leveraged to eliminate switch errors between already computed phasing blocks. This limits the room for improvements on the SER and SFR metrics. For the HR, however, we can see much larger improvements for some datasets. The HiFi readsets on chromosome 3 and 5 regions show the largest benefits, while the chromosome 4 region stays almost unaffected. In general, the HR still stays on a very high level despite the small tested regions.

Cutting the phasing on disconnected variants is a sensible choice for purely read-based approaches and can lead to lower HR values, e.g. for HiFi chromosome 4 region and Illumina chromosome 5 region. However, the N50 block size, which we normalized to the length of each tested region, decreases dramatically. The reason for that is probably large indels on the sequenced individual, which aggravate the correct read alignments and induce coverage gaps among the variants.

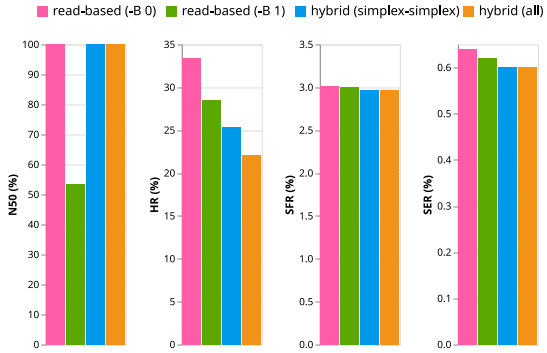
Sparse haplotypes as super reads

Another idea to utilize the sparse haplotypes is to interpret them as a set of so-called *super reads* that span the entire chromosome. For every phased variant, each of the p super reads contains the corresponding allele, while unphased variants result in a gap on all super reads. When running WHATSHAP POLYPHASE, we add the super reads to the readset $\mathcal{R}$ to incorporate the contained connectivity information. For the clustering stage (see Section 3.2), we require that all super reads have to be assigned to different clusters by manually inserting forbidden edges into the read graph. Instead of computing a fragmented clustering, all reads should rather be connected to their most similar super read and form much larger clusters that span the entire chromosome.

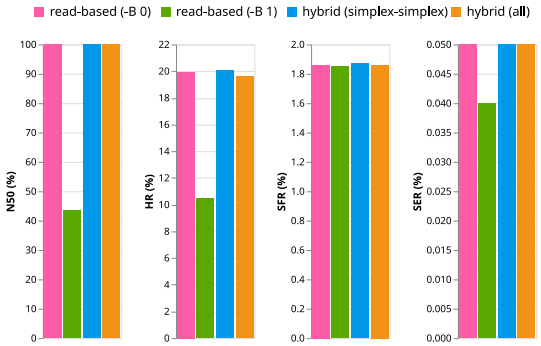
The advantage over the first method is the earlier inclusion of pre-phasing information to avoid errors that cannot be corrected anymore in the reordering stage. In practice, however,



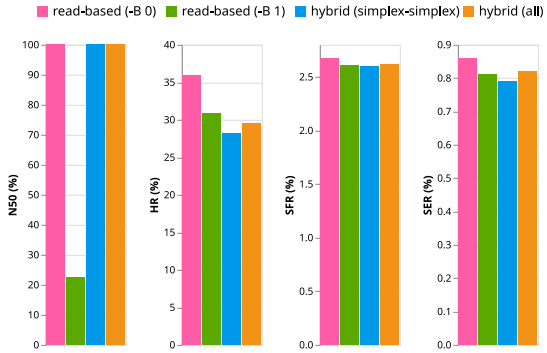
(a) Chromosome 3 region phased with HiFi reads



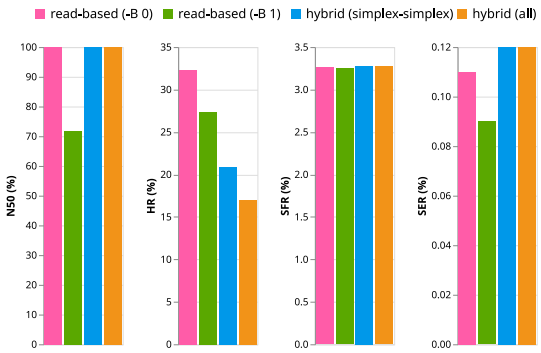
(b) Chromosome 3 region phased with Illumina reads



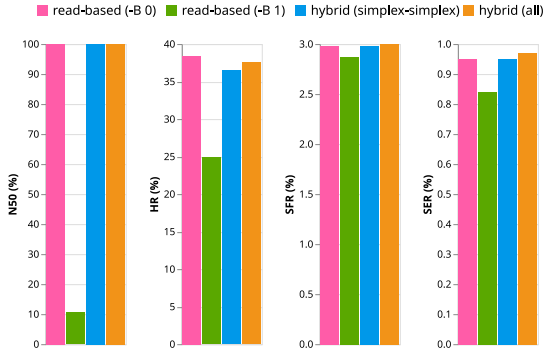
(c) Chromosome 4 region phased with HiFi reads



(d) Chromosome 4 region phased with Illumina reads



(e) Chromosome 5 region phased with HiFi reads



(f) Chromosome 5 region phased with Illumina reads

Figure 4.10: **Evaluation of hybrid phasing.** The plots show a comparison between pure read-based phasing and hybrid phasing that additionally uses pre-phasing information based on genetic phasing. The former was run without cut positions (`-B 0`) and with cuts whenever no read connects two consecutive variants (`-B 1`). The latter was run with pre-phasings containing only simplex-simplex variants and pre-phasings with all available variant types. N50 is given as a percentage of the length of the phased region.

the idea proved to scale very poorly with large readsets. The reason lies in the way the clustering algorithm works: Whenever an edge weight is updated in the read graph G, the icp and icf values for all incident non-zero edges have to be updated as well. The average number of non-zero neighbors for any node in G is close to the average coverage in $\mathcal{R}$ because only read pairs that overlap on at least two variant positions receive a non-zero score. The super reads represent hub nodes with degree $\mathcal{O}(|\mathcal{R}|)$ in the read graph because they overlap with almost every other read. This means that every edge update between a regular read r_1 and a super read r_2 requires an icp and icf update on every other edge that is incident to r_2. We ran the

cluster editing heuristic for the small selected regions defined in Section 4.3.1 but already ran into scalability issues with a more than a hundred-fold increase in running time compared to an instance without super reads. Since we expect the runtime increase to grow with larger regions based on the above observation, we decided that this implementation is not tractable to compute long-range phasings, which was its original purpose.

4.5 Discussion

Many discussion points are picked from the discussion in [16], but I added further explanations on the limitations to certain variant types and included the findings about hybrid phasing (integrating both genetic and read-based phasing) that were firstly described in this thesis. Some phrasings are borrowed from [16].

In this chapter, we presented our novel phasing algorithm WHATSHAP POLYPHASE-GENETIC which is based on genotype (or allele depth) information from a large offspring population of two parental samples. In contrast, the polyploid phasing method from Chapter 3 uses read information from a single sample directly and does neither utilize nor require data from additional individuals. The main advantage of WHATSHAP POLYPHASE-GENETIC is the ability to coherently phase variants over longer distances without depending on read connectivity, as we showed in our experiments in Section 4.3.1 that are only based on short reads. It can handle multiple genotype configurations that make up a large fraction of the present variants among the phased parents. In addition, our method is scalable to a whole chromosome using moderate computational resources.

4.5.1 Limitations

On the contrary, WHATSHAP POLYPHASE-GENETIC also faces several limitations. Even though we implemented support for multiple parental genotype patterns, it still has to omit a significant amount of variants compared to read-based phasers. This limitation is inherent to the statistical approach of the allele graph: Simply adding an implementation for all remaining genotype configurations will likely result in low phasing accuracy, as we already observed a significant rise in error rates in our experiments when adding simplex-simplex and duplex-nulliplex variants to the default setting of only utilizing simplex-nulliplex variants. The design choice to compute a maximum conflict-free assignment of marker alleles to haplotypes after the clustering step over computing a conflict-minimizing but complete assignment leads to more unphased variants. It is consistent with the choice of limiting the allele clustering to easy-to-phase variant types, but also prevents our algorithm from producing a complete phasing.

To circumvent the described issue, we combined the genetic phasing with our read-based phasing from Chapter 3. Even though we observed minor improvements in phasing quality when supporting the read-based method with the sparse haplotypes computed by WHATSHAP

POLYPHASE-GENETIC, the accuracy is overall much lower than for the genetic phasing alone, especially when using the short Illumina reads. We did not investigate these results any further, but the pre-phasing input for WHATSHAP POLYPHASE seems to only have little impact on the final phasing and is unable to repair any mistakes that might have been made in the first two stages of the algorithm.

Another limitation concerns our experiments themselves. We pointed out before that we used HiFi-based genome assemblies to identify regions that exhibit four distinct homologous contigs because there exists no gold-standard phasing for the sequenced potato samples we worked on. Since this effort only yielded three rather small regions of about 300kb each, the validity of our findings is quite limited.

A major issue in this context certainly is the high genetic diversity of the potato genome which is hard to represent with a linear reference genome. In any case, aligning a contig from hifiasm against the Solyntus reference genome produced hundreds of small alignments, some of them even scattered over different chromosomes. This is likely the result of structural variants, including large insertions and deletions, which have been revealed in a recent study by Sun et al. [51]. The fact that there are large insertions or deletions on single haplotypes – let alone more complex rearrangements – and thus not always p haplotypes present at each site, is not accounted for by current polyploid phasers, including the method described here.

4.5.2 Future work

The stated limitations open up opportunities to improve on our research in follow-up work: On the algorithmic side, the most interesting question is how read-based and genetic phasing can be combined more effectively. We already described an effort to include the sparse haplotypes from the genetic phasing as long super reads for the read-based phasing. One could either explore other ways to include the pre-phasing into earlier stages of WHATSHAP POLYPHASE that do not pose the same computational scalability issues or one could also develop a combined model that performs genetic and read-based phasing simultaneously.

On the validation side, a simulation study could provide sufficient proof for the phasing accuracy of our model over a larger distance, which is the main benefit of genetic phasing. Since the simulation allows control over all intermediately generated data, it enables a comparison between WHATSHAP POLYPHASE-GENETIC and the existing genetic phasers TriPoly and PopPoly. We note here that data simulation does not replace tests on real data, because there are a lot of unknown parameters in the simulation process, such as how to generate structural variants on the simulated haplotypes and what the recombination landscape looks like for the simulated species. As an alternative, the assemblies computed in [52] on the same offspring data but with additional HiFi reads for the parents could be translated into a phasing for the linear reference genome to validate the sparse phasings from our method.

Conclusions

Throughout this thesis, we reviewed and discussed many different approaches for reference-based haplotype phasing. In Chapter 2, we focused on the existing phasing tool WHATSHAP, which infers haplotypes for either diploid individuals (either singles or related trios) using aligned reads and an existing variant calling. Diploid haplotypes are (in most cases) complementary to each other because heterozygosity directly implies different alleles on each haplotype. Thus, one haplotype can be derived from the other.

For single individuals, WHATSHAP solves the commonly used MEC model to optimality. For related individuals, it solves the generalized PedMEC model [13], which adds additional constraints based on the rules of Mendelian inheritance. To counteract the limitation of WHATSHAP of exponential runtime scaling with coverage among all individuals, only a subset of the input data can be used in the process. As a result, the average haploid coverage of each individual becomes quite low for pedigree phasing. We attempted to tackle this problem with a heuristic approach for the PedMEC model that is able to process higher coverages at the expense of losing optimality. We compared the heuristic to the exact solver in terms of phasing quality and runtime. As test dataset, we used a human sample with gold-standard haplotypes and real reads for the sample itself and its parents available. While the heuristic was able to process each instance and phased slightly more variants, it rarely surpassed the exact algorithm in phasing quality using the additional data. Nevertheless, it proved to be competitive in terms of computational resources. We also observed some anomalies in the experiments when passing datasets with high coverage to the PedMEC solver. This issue and the fact there the heuristic generally phases additional variants of potentially low quality over the exact algorithm leaves room for further development in the future.

Moving from diploid to polyploid phasing, we discussed the arising challenges for polyploid genomes in Chapter 3. We pointed out that the MEC model is unsuited for the polyploid case because it does not consider collapsed regions and uniform haplotype coverage. To address these challenges, we presented WHATSHAP POLYPHASE as a novel approach for single-individual, read-based polyploid phasing. We have already published several of these ideas before in a prior publication [14] but added several new ones in this thesis to improve the model. We presented the most recent version in detail and pointed out differences to the published version. For evaluation, we repeated our experiments on an artificial tetraploid human from our previous work [14] with both the old and new versions of WHATSHAP POLYPHASE. We

again compared our work to H-PoP-G [40] – in its newest version – and also included a newer tool called FLOPP [15], which was published well after our initial publication. Surprisingly, the changes of the new version of WHATSHAP POLYPHASE over the old one showed performance regressions regarding phasing quality, whose specific cause we were unable to pin down fully. However, on the datasets generated by Shaw and Yu [15] and HiFi reads from a real potato sample, the adjustments of the revised algorithm resulted in a clear improvement over the old one. Compared to the other algorithms, a general advantage of (both versions of) WHATSHAP POLYPHASE remains to be the flexible cut position policy, enabling a trade-off between phase block length and phasing accuracy.

Even though WHATSHAP POLYPHASE offers better performance in SER and SFR metrics than H-PoP-G and FLOPP when considering the better out of old and new version for each dataset, the conducted tests revealed room for improvement in several areas. Large phasing blocks suffer from critical switch errors that severely degrade the performance on the HR metric, while H-PoP-G and FLOPP seem less susceptible to this issue. In combination with the adjustable block cut strategy, it would be highly desirable to identify the cause of these critical switches and develop better criteria for phase block separation, such that the blocks become longer but maintain their high accuracy. Another issue is the substantial decline in switch errors when dropping the requirement to match the input genotype from the VCF file. Since genotype enforcement is a late step in our algorithm, genotype divergences before this point could hint at potential error hotspots and a more sophisticated enforcement model could be an opportunity for further quality improvements.

In addition to the single-individual WHATSHAP POLYPHASE, we adapted pedigree phasing for the polyploid case in Chapter 4 and presented a second method, called WHATSHAP POLYPHASE-GENETIC. In contrast to the former, our latter method leverages genotype and allele depth information instead of reads. It requires data from two parental samples and a large offspring population, and is thus not suitable for single individuals. Moreover, it is only able to phase variants with certain genotype patterns among the parents due to the insufficient statistical signal on the unphased patterns. For a couple of small selected regions of the potato genome, we were able to infer four distinct homologous contigs from a HiFi assembly. These contigs served as a ground-truth phasing for these regions and, as a proof-of-concept for the accuracy of our method, we showed that these contig-based phasings are close to what WHATSHAP POLYPHASE-GENETIC computes for the same regions. We also demonstrated the ability of our method to scale to full chromosomes, although we were unable to validate its phasing quality due to a missing reference phasing. In addition, we combined our two algorithms into a hybrid method, where the sparse genetic phasing is used as a pre-phasing for WHATSHAP POLYPHASE to reorder haplotype blocks and the accuracy for long phasing blocks. However, the performed tests only revealed subtle improvements compared to purely read-based phasing. We assume that the reordering stage is unable to correct errors made in the previous stages and suggest as future research either integrating the pre-phasing already in

earlier stages of WHATSHAP POLYPHASE or developing a combined model that uses both reads and pedigree information simultaneously.

Validating polyploid phasing methods proved to be a difficult task because haplotype-resolved samples suitable as gold standards are very rare. Many published assemblies of plant genomes are based on diploid assemblies by either using the allopolyploid nature of the species [103, 104, 105, 106] or sequencing pollen that only contain half a haplotype set [51]. This results in many polyploid phasing algorithms (including our own) being validated on simulated data, which tends to be more benevolent to process than real data because structural variants or repetitive regions are not well-studied enough to incorporate them in the simulation process in a representative way. Since applicability to real-world datasets is an important feature of novel phasing tools, it is difficult to judge the performance of a tool from simulated data alone. The initial version of WHATSHAP POLYPHASE, for instance, performed well on our artificial tetraploid human dataset but showed severe issues on unseen simulated datasets from other studies. We, therefore, believe that the research field of polyploid phasing would greatly benefit from high-quality, haplotype-resolved assemblies of autopolyploid samples that serve as an accurate benchmark for existing and future tools.

On a larger note, the general approach of reference-based phasing should also be critically discussed; in human genomics, there has been a tremendous effort to create a high-quality linear representation of the human genome, resulting in the latest CHM13 reference by the Telomere-to-Telomere consortium [110]. Polyploid plant genomes, however, contain much more diversity and structural variants, as pointed out in the discussion in Section 4.5. This considerably aggravates the construction of good reference genomes and the projection of phased individuals to a linear genome space. In our studies of the Altus and Colomba potato cultivars we experienced several issues that we believe are related to the limitations of linear references.

This observation should motivate future research on polyploid phasing to shift the focus to either non-linear references or reference-free approaches. In the discussion in Section 3.6, we already proposed the use of reference graphs as a more powerful means to express genetic variation. This representation has gained a lot of attention in the recent past in the context of pangenomics and the need to store the genomic diversity of large populations in a single structure [111]. The methods presented in this thesis are not directly applicable to such graphs – as they require a linear coordinate system – but some of the core ideas might be adjustable to non-linear structures for further research. Assembly-based approaches are potentially even more powerful, as they do not rely at all on previous knowledge about the species to phase. In a recent study by Serra Mari et al. [52], we successfully computed haplotype-resolved assemblies for the same Altus cultivar that we used for evaluation in Chapter 4. Instead of a reference genome, the algorithm requires data from multiple sequencing technologies and multiple individuals, which is a lot of effort for a single phasing.

In summary, reference-based phasing methods will likely stay more limited in what variation they can resolve on polyploid species compared to diploid phasing on human genomes.

However, this does not generally render reference-based phasing methods unsuitable for polyploid genomes in comparison to reference-free methods, because (i) the stored information in the reference allows for a more cost-efficient way to generate phasings and (ii) it keeps the compatibility to many existing tools in bioinformatics (e.g. variant callers) that are built on reference genomes as well. As a result, we believe that the two paradigms form complementary approaches for the same problem and will both profit from one another regarding future research advances.

Bibliography

[1] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular biology of the cell*. Volume 5. Garland Science, 2008. ISBN: 9780815341055.

[2] R. Tewhey, V. Bansal, A. Torkamani, E. J. Topol, and N. J. Schork. "The importance of phase information for human genomics". In: *Nature Reviews Genetics* 12.3 (Feb. 2011), pages 215–223. DOI: 10.1038/nrg2950.

[3] G. Glusman, H. C. Cox, and J. C. Roach. "Whole-genome haplotyping approaches and genomic medicine". In: *Genome Medicine* 6.9 (Sept. 2014). 73. DOI: 10.1186/s13073-014-0073-7.

[4] D. J. Lawson, G. Hellenthal, S. Myers, and D. Falush. "Inference of Population Structure using Dense Haplotype Data". In: *PLOS Genetics* 8.1 (Jan. 2012), pages 1–16. DOI: 10.1371/journal.pgen.1002453.

[5] P. C. Sabeti, P. Varilly, B. Fry, J. Lohmueller, E. Hostetter, C. Cotsapas, X. Xie, E. H. Byrne, S. A. McCarroll, R. Gaudet, S. F. Schaffner, E. S. Lander, and T. I. H. Consortium. "Genome-wide detection and characterization of positive selection in human populations". In: *Nature* 449.7164 (Oct. 2007), pages 913–918. DOI: 10.1038/nature06250.

[6] E. Ukkonen. "Finding Founder Sequences from a Set of Recombinants". In: *Algorithms in Bioinformatics*. Edited by R. Guigó and D. Gusfield. Springer Berlin Heidelberg, 2002, pages 277–286. ISBN: 978-3-540-45784-8. DOI: 10.1007/3-540-45784-4_21.

[7] K. Bonnet, T. Marschall, and D. Doerr. "Constructing founder sets under allelic and non-allelic homologous recombination". In: *Algorithms for Molecular Biology* 18.1 (Sept. 2023), page 15. DOI: 10.1186/s13015-023-00241-3.

[8] L. Xie, X. Gong, K. Yang, Y. Huang, S. Zhang, L. Shen, Y. Sun, D. Wu, C. Ye, Q.-H. Zhu, and L. Fan. "Technology-enabled great leap in deciphering plant genomes". In: *Nature Plants* (Mar. 2024). DOI: 10.1038/s41477-024-01655-6.

[9] J. Yang, M.-H. Moeinzadeh, H. Kuhl, J. Helmuth, P. Xiao, S. Haas, G. Liu, J. Zheng, Z. Sun, W. Fan, G. Deng, H. Wang, F. Hu, S. Zhao, A. R. Fernie, S. Boerno, B. Timmermann, P. Zhang, and M. Vingron. "Haplotype-resolved sweet potato genome traces back its hexaploidization history". In: *Nature Plants* (Aug. 2017). DOI: 10.1038/s41477-017-0002-z.

[10] R. G. F. Visser, C. W. B. Bachem, T. Borm, J. de Boer, H. J. van Eck, R. Finkers, G. van der Linden, C. A. Maliepaard, J G A M, R. Voorrips, P. Vos, and A. M. A. Wolters. "Possibilities and Challenges of the Potato Genome Sequence". In: *Potato Res.* 57.3-4 (Dec. 2014), pages 327–330. DOI: 10.1007/s11540-015-9282-8.

[11] K.-T. Li, M. Moulin, N. Mangel, M. Albersen, N. M. Verhoeven-Duif, Q. Ma, P. Zhang, T. B. Fitzpatrick, W. Gruissem, and H. Vanderschuren. "Increased bioavailable vitamin B6 in field-grown transgenic cassava for dietary sufficiency". In: *Nature Biotechnology* 33 (Oct. 2015), pages 1029–1032. DOI: 10.1038/nbt.3318.

[12] M. Patterson, T. Marschall, N. Pisanti, L. van Iersel, L. Stougie, G. W. Klau, and A. Schönhuth. "WhatsHap: Weighted Haplotype Assembly for Future-Generation Sequencing Reads". In: *Journal of Computational Biology* 22.6 (June 2015), pages 498–509. DOI: 10.1089/cmb.2014.0157.

[13] S. Garg, M. Martin, and T. Marschall. "Read-based phasing of related individuals". In: *Bioinformatics* 32.12 (June 2016), pages i234–i242. DOI: 10.1093/bioinformatics/btw276.

[14] S. Schrinner, R. Mari, J. Ebler, M. Rautiainen, L. Seillier, J. Reimer, B. Usadel, T. Marschall, and G. Klau. "Haplotype threading: accurate polyploid phasing from long reads". In: *Genome biology* 21 (Sept. 2020). 252. DOI: 10.1186/s13059-020-02158-1.

[15] J. Shaw and Y. W. Yu. "flopp: Extremely Fast Long-Read Polyploid Haplotype Phasing by Uniform Tree Partitioning". In: *Journal of Computational Biology* 29.2 (2022), pages 195–211. DOI: 10.1089/cmb.2021.0436.

[16] S. Schrinner, R. Serra Mari, R. Finkers, P. Arens, B. Usadel, T. Marschall, and G. W. Klau. "Genetic polyploid phasing from low-depth progeny samples". In: *iScience* 25.6 (2022), page 104461. DOI: 10.1016/j.isci.2022.104461.

[17] F. Sanger, S. Nicklen, and A. R. Coulson. "DNA sequencing with chain-terminating inhibitors". In: *Proceedings of the National Academy of Sciences of the United States of America* 74.12 (Dec. 1977), pages 5463–5467. DOI: 10.1073/pnas.74.12.5463.

[18] A. M. Maxam and W. Gilbert. "A new method for sequencing DNA." In: *Proceedings of the National Academy of Sciences of the United States of America* 74.2 (Feb. 1977), pages 560–564. DOI: 10.1073/pnas.74.2.560.

[19] J. Shendure, S. Balasubramanian, G. M. Church, W. Gilbert, J. Rogers, J. A. Schloss, and R. H. Waterston. "DNA sequencing at 40: past, present and future". In: *Nature* 550.7676 (Oct. 2017), pages 345–353. DOI: 10.1038/nature24286.

[20] W. R. McCombie, J. D. McPherson, and E. R. Mardis. "Next-generation sequencing technologies". In: *Cold Spring Harb. Perspect. Med.* 9.11 (Nov. 2019). DOI: 10.1101/cshperspect.a036798.

[21] M. M. Mohammadi and O. Bavi. “DNA sequencing: an overview of solid-state and biological nanopore-based methods”. In: *Biophysical Reviews* 14.1 (Feb. 2022), pages 99–110. DOI: 10.1007/s12551-021-00857-y.

[22] Illumina Inc. *An introduction to Next-Generation Sequencing Technology*. URL: https://www.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf. Accessed: 2024-03-29.

[23] G. A. Logsdon, M. R. Vollger, and E. E. Eichler. “Long-read human genome sequencing and its applications”. In: *Nature Reviews Genetics* 21.10 (Oct. 2020), pages 597–614. DOI: 10.1038/s41576-020-0236-x.

[24] Illumina Inc. *Specifications for the NextSeq 550 System*. URL: https://www.illumina.com/systems/sequencing-platforms/nextseq/specifications.html. Accessed: 2024-03-29.

[25] F. Pfeiffer, C. Gröber, M. Blank, K. Händler, M. Beyer, J. L. Schultze, and G. Mayer. “Systematic evaluation of error rates and causes in short samples in next-generation sequencing”. In: *Scientific Reports* 8.1 (July 2018). 10950. DOI: 10.1038/s41598-018-29325-6.

[26] N. Stoler and A. Nekrutenko. “Sequencing error profiles of Illumina sequencing instruments”. In: *NAR Genomics and Bioinformatics* 3.1 (Mar. 2021). lqab019. DOI: 10.1093/nargab/lqab019.

[27] J. L. Weirather, M. de Cesare, Y. Wang, P. Piazza, V. Sebastiano, X.-J. Wang, D. Buck, and K. F. Au. “Comprehensive comparison of Pacific Biosciences and Oxford Nanopore Technologies and their applications to transcriptome analysis [version 2; peer review: 2 approved]”. In: *F1000Research* 6.100 (June 2017). DOI: 10.12688/f1000research.10571.2.

[28] A. M. Wenger, P. Peluso, W. J. Rowell, P.-C. Chang, R. J. Hall, G. T. Concepcion, J. Ebler, A. Fungtammasan, A. Kolesnikov, N. D. Olson, A. Töpfer, M. Alonge, M. Mahmoud, Y. Qian, C.-S. Chin, A. M. Phillippy, M. C. Schatz, G. Myers, M. A. DePristo, J. Ruan, T. Marschall, F. J. Sedlazeck, J. M. Zook, H. Li, S. Koren, A. Carroll, D. R. Rank, and M. W. Hunkapiller. “Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome”. In: *Nature Biotechnology* 37.10 (Oct. 2019), pages 1155–1162. DOI: 10.1038/s41587-019-0217-9.

[29] S. B. Needleman and C. D. Wunsch. “A general method applicable to the search for similarities in the amino acid sequence of two proteins”. In: *Journal of Molecular Biology* 48.3 (1970), pages 443–453. DOI: 10.1016/0022-2836(70)90057-4.

[30] T. Smith and M. Waterman. “Identification of common molecular subsequences”. In: *Journal of Molecular Biology* 147.1 (1981), pages 195–197. DOI: 10.1016/0022-2836(81)90087-5.

[31] H. Li. “Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM”. In: *ArXiv* 1303 (Mar. 2013). DOI: 10.48550/arXiv.1303.3997.

[32] H. Li. “Minimap2: pairwise alignment for nucleotide sequences”. In: *Bioinformatics* 34.18 (May 2018), pages 3094–3100. DOI: 10.1093/bioinformatics/bty191.

[33] The 1000 Genomes Project Consortium. “A global reference for human genetic variation”. In: *Nature* 526.7571 (Oct. 2015), pages 68–74. DOI: 10.1038/nature15393.

[34] J. G. A. M. L. Uitdewilligen, A.-M. A. Wolters, B. B. Dhoop, T. J. A. Borm, R. G. F. Visser, and H. J. van Eck. “A Next-Generation Sequencing Method for Genotyping-by-Sequencing of Highly Heterozygous Autotetraploid Potato”. In: *PLOS ONE* 8.5 (May 2013), pages 1–14. DOI: 10.1371/journal.pone.0062355.

[35] B. Paten, A. M. Novak, J. M. Eizenga, and E. Garrison. “Genome graphs and the evolution of genome inference”. In: *Genome Research* 27.5 (May 2017), pages 665–676. DOI: 10.1101/gr.214155.116.

[36] E. Garrison and G. Marth. “Haplotype-based variant detection from short-read sequencing”. In: *arXiv* 1207 (July 2012). DOI: 10.48550/arXiv.1207.3907.

[37] R. Poplin, V. Ruano-Rubio, M. A. DePristo, T. J. Fennell, M. O. Carneiro, G. A. V. der Auwera, D. E. Kling, L. D. Gauthier, A. Levy-Moonshine, D. Roazen, K. Shakir, J. Thibault, S. Chandran, C. Whelan, M. Lek, S. Gabriel, M. J. Daly, B. Neale, D. G. MacArthur, and E. Banks. “Scaling accurate genetic variant discovery to tens of thousands of samples”. In: *bioRxiv* (2018). DOI: 10.1101/201178.

[38] J. Köster, L. J. Dijkstra, T. Marschall, and A. Schönhuth. “Varlociraptor: enhancing sensitivity and controlling false discovery rate in somatic indel discovery”. In: *Genome Biology* 21.1 (Apr. 2020). 98. DOI: 10.1186/s13059-020-01993-6.

[39] G. W. Klau and T. Marschall. “A Guided Tour to Computational Haplotyping”. In: *Unveiling Dynamics and Complexity*. Volume 10307. Lecture Notes in Computer Science. Cham: Springer, June 2017, pages 50–63. DOI: 10.1007/978-3-319-58741-7_6.

[40] M. Xie, Q. Wu, J. Wang, and T. Jiang. “H-PoP and H-PoPG: heuristic partitioning algorithms for single individual haplotyping of polyploids”. In: *Bioinformatics* 32.24 (Aug. 2016), pages 3735–3744. DOI: 10.1093/bioinformatics/btw537.

[41] J. C. Roach, G. Glusman, R. Hubley, S. Z. Montsaroff, A. K. Holloway, D. E. Mauldin, D. Srivastava, V. Garg, K. S. Pollard, D. J. Galas, L. Hood, and A. F. A. Smit. “Chromosomal Haplotypes by Genetic Phasing of Human Families”. In: *The American Journal of Human Genetics* 89.3 (Sept. 2011), pages 382–397. DOI: 10.1016/j.ajhg.2011.07.023.

[42] E. Motazedi, D. de Ridder, R. Finkers, S. Baldwin, S. Thomson, K. Monaghan, and C. Maliepaard. “TriPoly: haplotype estimation for polyploids using sequencing data of related individuals”. In: *Bioinformatics* 34.22 (June 2018), pages 3864–3872. DOI: 10.1093/bioinformatics/bty442.

[43] E. Motazedi, C. Maliepaard, R. Finkers, R. Visser, and D. de Ridder. “Family-Based Haplotype Estimation and Allele Dosage Correction for Polyploids Using Short Sequence Reads”. In: *Frontiers in Genetics* 10 (2019). DOI: 10.3389/fgene.2019.00335.

[44] S. R. Browning and B. L. Browning. “Haplotype phasing: existing methods and new developments”. In: *Nature Reviews Genetics* 12.10 (Oct. 2011), pages 703–714. DOI: 10.1038/nrg3054.

[45] B. L. Browning, X. Tian, Y. Zhou, and S. R. Browning. “Fast two-stage phasing of large-scale sequence data”. In: *The American Journal of Human Genetics* 108.10 (2021), pages 1880–1890. DOI: https://doi.org/10.1016/j.ajhg.2021.08.005.

[46] P.-R. Loh, P. Danecek, P. F. Palamara, C. Fuchsberger, Y. A Reshef, H. K Finucane, S. Schoenherr, L. Forer, S. McCarthy, G. R. Abecasis, R. Durbin, and A. L Price. “Reference-based phasing using the Haplotype Reference Consortium panel”. In: *Nature Genetics* 48.11 (Nov. 2016), pages 1443–1448. DOI: 10.1038/ng.3679.

[47] R. J. Hofmeister, D. M. Ribeiro, S. Rubinacci, and O. Delaneau. “Accurate rare variant phasing of whole-genome and whole-exome sequencing data in the UK Biobank”. In: *Nature Genetics* 55.7 (July 2023), pages 1243–1249. DOI: 10.1038/s41588-023-01415-w.

[48] M. J. P. Chaisson, A. D. Sanders, X. Zhao, A. Malhotra, D. Porubsky, T. Rausch, E. J. Gardner, O. L. Rodriguez, L. Guo, R. L. Collins, X. Fan, J. Wen, R. E. Handsaker, S. Fairley, Z. N. Kronenberg, X. Kong, F. Hormozdiari, D. Lee, A. M. Wenger, A. R. Hastie, D. Antaki, T. Anantharaman, P. A. Audano, H. Brand, S. Cantsilieris, H. Cao, E. Cerveira, C. Chen, X. Chen, C.-S. Chin, Z. Chong, N. T. Chuang, C. C. Lambert, D. M. Church, L. Clarke, A. Farrell, J. Flores, T. Galeev, D. U. Gorkin, M. Gujral, V. Guryev, W. H. Heaton, J. Korlach, S. Kumar, J. Y. Kwon, E. T. Lam, J. E. Lee, J. Lee, W.-P. Lee, S. P. Lee, S. Li, P. Marks, K. Viaud-Martinez, S. Meiers, K. M. Munson, F. C. P. Navarro, B. J. Nelson, C. Nodzak, A. Noor, S. Kyriazopoulou-Panagiotopoulou, A. W. C. Pang, Y. Qiu, G. Rosanio, M. Ryan, A. Stütz, D. C. J. Spierings, A. Ward, A. E. Welch, M. Xiao, W. Xu, C. Zhang, Q. Zhu, X. Zheng-Bradley, E. Lowy, S. Yakneen, S. McCarroll, G. Jun, L. Ding, C. L. Koh, B. Ren, P. Flicek, K. Chen, M. B. Gerstein, P.-Y. Kwok, P. M. Lansdorp, G. T. Marth, J. Sebat, X. Shi, A. Bashir, K. Ye, S. E. Devine, M. E. Talkowski, R. E. Mills, T. Marschall, J. O. Korbel, E. E. Eichler, and C. Lee. “Multi-platform discovery of haplotype-resolved structural variation in human genomes”. In: *Nature Communications* 10.1 (Apr. 2019). 1784. DOI: 10.1038/s41467-018-08148-z.

[49] P. Ebert, P. A. Audano, Q. Zhu, B. Rodriguez-Martin, D. Porubsky, M. J. Bonder, A. Sulovari, J. Ebler, W. Zhou, R. S. Mari, F. Yilmaz, X. Zhao, P. Hsieh, J. Lee, S. Kumar, J. Lin, T. Rausch, Y. Chen, J. Ren, M. Santamarina, W. Höps, H. Ashraf, N. T. Chuang, X. Yang, K. M. Munson, A. P. Lewis, S. Fairley, L. J. Tallon, W. E. Clarke, A. O. Basile, M. Byrska-Bishop, A. Corvelo, U. S. Evani, T.-Y. Lu, M. J. P. Chaisson, J. Chen, C. Li,

H. Brand, A. M. Wenger, M. Ghareghani, W. T. Harvey, B. Raeder, P. Hasenfeld, A. A. Regier, H. J. Abel, I. M. Hall, P. Flicek, O. Stegle, M. B. Gerstein, J. M. C. Tubio, Z. Mu, Y. I. Li, X. Shi, A. R. Hastie, K. Ye, Z. Chong, A. D. Sanders, M. C. Zody, M. E. Talkowski, R. E. Mills, S. E. Devine, C. Lee, J. O. Korbel, T. Marschall, and E. E. Eichler. "Haplotype-resolved diverse human genomes and integrated analysis of structural variation". In: *Science* 372.6537 (2021). DOI: 10.1126/science.abf7117.

[50] N. van Lieshout, A. van der Burgt, M. E. de Vries, M. ter Maat, D. Eickholt, D. Esselink, M. P. W. van Kaauwen, L. P. Kodde, R. G. F. Visser, P. Lindhout, and R. Finkers. "Solyntus, the new highly contiguous reference genome for potato (*Solanum tuberosum*)". In: *G3 (Bethesda)* 10.10 (Oct. 2020), pages 3489–3495. DOI: 10.1534/g3.120.401550.

[51] H. Sun, W.-B. Jiao, K. Krause, J. A. Campoy, M. Goel, K. Folz-Donahue, C. Kukat, B. Huettel, and K. Schneeberger. "Chromosome-scale and haplotype-resolved genome assembly of a tetraploid potato cultivar". In: *Nature Genetics* 54.3 (Mar. 2022), pages 342–348. DOI: 10.1038/s41588-022-01015-0.

[52] R. Serra Mari, S. Schrinner, R. Finkers, F. M. R. Ziegler, P. Arens, M. H.-W. Schmidt, B. Usadel, G. W. Klau, and T. Marschall. "Haplotype-resolved assembly of a tetraploid potato genome using long reads and low-depth offspring data". In: *Genome Biology* 25.1 (Jan. 2024). DOI: 10.1186/s13059-023-03160-z.

[53] X. Zhang, R. Wu, Y. Wang, J. Yu, and H. Tang. "Unzipping haplotypes in diploid and polyploid genomes". In: *Computational and Structural Biotechnology Journal* 18 (2020), pages 66–72. DOI: 10.1016/j.csbj.2019.11.011.

[54] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy. "Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation". In: *Genome Research* 27.5 (Mar. 2017), pages 722–736. DOI: 10.1101/gr.215087.116.

[55] M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner. "Assembly of long, error-prone reads using repeat graphs". In: *Nature Biotechnology* 37.5 (May 2019), pages 540–546. DOI: 10.1038/s41587-019-0072-8.

[56] H. Cheng, G. T. Concepcion, X. Feng, H. Zhang, and H. Li. "Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm". In: *Nature Methods* 18.2 (Feb. 2021), pages 170–175. DOI: 10.1038/s41592-020-01056-5.

[57] S. Koren, A. Rhie, B. P. Walenz, A. T. Dilthey, D. M. Bickhart, S. B. Kingan, S. Hiendleder, J. L. Williams, T. P. L. Smith, and A. M. Phillippy. "De novo assembly of haplotype-resolved genomes with trio binning". In: *Nature Biotechnology* 36.12 (Dec. 2018), pages 1174–1182. DOI: 10.1038/nbt.4277.

[58] Z. N. Kronenberg, A. Rhie, S. Koren, G. T. Concepcion, P. Peluso, K. M. Munson, D. Porubsky, K. Kuhn, K. A. Mueller, W. Y. Low, S. Hiendleder, O. Fedrigo, I. Liachko, R. J. Hall, A. M. Phillippy, E. E. Eichler, J. L. Williams, T. P. L. Smith, E. D. Jarvis, S. T. Sullivan, and S. B. Kingan. "Extended haplotype-phasing of long-read de novo genome assemblies using Hi-C". In: *Nature Communications* 12.1 (Apr. 2021), page 1935. DOI: 10.1038/s41467-020-20536-y.

[59] E. Motazedi, R. Finkers, C. Maliepaard, and D. de Ridder. "Exploiting next-generation sequencing to solve the haplotyping puzzle in polyploids: a simulation study". In: *Briefings in Bioinformatics* 19.3 (May 2018), pages 387–403. DOI: 10.1093/bib/bbw126.

[60] E. Berger, D. Yorukoglu, J. Peng, and B. Berger. "HapTree: A Novel Bayesian Framework for Single Individual Polyplotyping Using NGS Data". In: *PLOS Computational Biology* 10.3 (Mar. 2014), pages 1–10. DOI: 10.1371/journal.pcbi.1003502.

[61] National Center for Biotechnology Information and U.S. National Library of Medicine. *File Format Guide*. URL: https://www.ncbi.nlm.nih.gov/sra/docs/submitformats/. Accessed: 2024-03-16.

[62] National Center for Biotechnology Information and U.S. National Library of Medicine. *BLAST Topics*. URL: https://blast.ncbi.nlm.nih.gov/doc/blast-topics/. Accessed: 2024-03-16.

[63] Illumina Inc. *FASTQ files explained*. URL: https://knowledge.illumina.com/software/general/software-general-reference_material-list/000002211. Accessed: 2024-03-16.

[64] Samtools. *The Variant Call Format Specification, VCFv4.3 and BCFv2.2*. URL: https://samtools.github.io/hts-specs/VCFv4.3.pdf. Accessed: 2024-03-16.

[65] R. Lippert, R. Schwartz, G. Lancia, and S. Istrail. "Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem". In: *Briefings in Bioinformatics* 3.1 (Mar. 2002), pages 23–31. DOI: 10.1093/bib/3.1.23.

[66] R. Cilibrasi, L. van Iersel, S. Kelk, and J. Tromp. "On the Complexity of Several Haplotyping Problems". In: *Algorithms in Bioinformatics*. Edited by R. Casadio and G. Myers. Springer Berlin Heidelberg, 2005, pages 128–139. ISBN: 978-3-540-31812-5. DOI: 10.1007/11557067_11.

[67] R.-S. Wang, L.-Y. Wu, Z.-P. Li, and X.-S. Zhang. "Haplotype reconstruction from SNP fragments by minimum error correction". In: *Bioinformatics* 21.10 (Feb. 2005), pages 2456–2462. DOI: 10.1093/bioinformatics/bti352.

[68] V. Bansal and V. Bafna. "HapCUT: an efficient and accurate algorithm for the haplotype assembly problem". In: *Bioinformatics* 24.16 (Aug. 2008), pages i153–i159. DOI: 10.1093/bioinformatics/btn298.

[69] D. He, A. Choi, K. Pipatsrisawat, A. Darwiche, and E. Eskin. “Optimal algorithms for haplotype assembly from whole-genome sequence data”. In: *Bioinformatics* 26.12 (June 2010), pages i183–i190. DOI: 10.1093/bioinformatics/btq215.

[70] Z.-Z. Chen, F. Deng, and L. Wang. “Exact algorithms for haplotype assembly from whole-genome sequence data”. In: *Bioinformatics* 29.16 (June 2013), pages 1938–1945. DOI: 10.1093/bioinformatics/btt349.

[71] M. Etemadi Maryam and Bagherian, Z.-Z. Chen, and L. Wang. “Better ILP models for haplotype assembly”. In: *BMC Bioinformatics* 19.52 (Feb. 2018). DOI: 10.1186/s12859-018-2012-x.

[72] J. M. Zook, D. Catoe, J. McDaniel, L. Vang, N. Spies, A. Sidow, Z. Weng, Y. Liu, C. E. Mason, N. Alexander, E. Henaff, A. B. McIntyre, D. Chandramohan, F. Chen, E. Jaeger, A. Moshrefi, K. Pham, W. Stedman, T. Liang, M. Saghbini, Z. Dzakula, A. Hastie, H. Cao, G. Deikus, E. Schadt, R. Sebra, A. Bashir, R. M. Truty, C. C. Chang, N. Gulbahce, K. Zhao, S. Ghosh, F. Hyland, Y. Fu, M. Chaisson, C. Xiao, J. Trow, S. T. Sherry, A. W. Zaranek, M. Ball, J. Bobe, P. Estep, G. M. Church, P. Marks, S. Kyriazopoulou-Panagiotopoulou, G. X. Zheng, M. Schnall-Levin, H. S. Ordonez, P. A. Mudivarti, K. Giorda, Y. Sheng, K. B. Rypdal, and M. Salit. “Extensive sequencing of seven human genomes to characterize benchmark reference materials”. In: *Scientific Data* 3.1 (June 2016). 160025. DOI: 10.1038/sdata.2016.25.

[73] O. Delaneau, J. Marchini, and The 1000 Genomes Project Consortium. “Integrating sequence and array data to create an improved 1000 Genomes Project haplotype reference panel”. In: *Nature Communications* 5.1 (June 2014). 3934. DOI: 10.1038/ncomms4934.

[74] J. Wagner, N. D. Olson, L. Harris, Z. Khan, J. Farek, M. Mahmoud, A. Stankovic, V. Kovacevic, B. Yoo, N. Miller, J. A. Rosenfeld, B. Ni, S. Zarate, M. Kirsche, S. Aganezov, M. C. Schatz, G. Narzisi, M. Byrska-Bishop, W. Clarke, U. S. Evani, C. Markello, K. Shafin, X. Zhou, A. Sidow, V. Bansal, P. Ebert, T. Marschall, P. Lansdorp, V. Hanlon, C.-A. Mattsson, A. M. Barrio, I. T. Fiddes, C. Xiao, A. Fungtammasan, C.-S. Chin, A. M. Wenger, W. J. Rowell, F. J. Sedlazeck, A. Carroll, M. Salit, and J. M. Zook. “Benchmarking challenging small variants with linked and long reads”. In: *Cell Genomics* 2.5 (2022). 100128. DOI: 10.1016/j.xgen.2022.100128.

[75] M. Rautiainen, S. Nurk, B. P. Walenz, G. A. Logsdon, D. Porubsky, A. Rhie, E. E. Eichler, A. M. Phillippy, and S. Koren. “Telomere-to-telomere assembly of diploid chromosomes with Verkko”. In: *Nature Biotechnology* 41.10 (Oct. 2023), pages 1474–1482. DOI: 10.1038/s41587-023-01662-6.

[76] F. Mölder, K. Jablonski, B. Letcher, M. Hall, C. Tomkins-Tinch, V. Sochat, J. Forster, S. Lee, S. Twardziok, A. Kanitz, A. Wilm, M. Holtgrewe, S. Rahmann, S. Nahnsen,

and J. Köster. “Sustainable data analysis with Snakemake [version 2; peer review: 2 approved]”. In: *F1000Research* 33 (2021). DOI: 10.12688/f1000research.29032.2.

[77] D. Aguiar and S. Istrail. “Haplotype assembly in polyploid genomes and identical by descent shared tracts”. In: *Bioinformatics* 29.13 (July 2013), pages i352–i360. DOI: 10.1093/bioinformatics/btt213.

[78] D. Aguiar and S. Istrail. “HapCompass: a fast cycle basis algorithm for accurate haplotype assembly of sequence data”. In: *Journal of Computational Biology* 19.6 (June 2012), pages 577–590. DOI: 10.1089/cmb.2012.0084.

[79] S. Das and H. Vikalo. “SDhaP: haplotype assembly for diploids and polyploids via semi-definite programming”. In: *BMC Genomics* 16 (Apr. 2015), page 260. DOI: 10.1186/s12864-015-1408-5.

[80] D. He, S. Saha, R. Finkers, and L. Parida. “Efficient algorithms for polyploid haplotype phasing”. In: *BMC Genomics* 19.Suppl 2 (May 2018). 110. DOI: 10.1186/s12864-018-4464-9.

[81] M. J. P. Chaisson, S. Mukherjee, S. Kannan, and E. E. Eichler. “Resolving multicopy duplications de novo using polyploid phasing”. In: *Research in Computational Molecular Biology* 10229 (May 2017), pages 117–133. DOI: 10.1007/978-3-319-56970-3_8.

[82] C. Cai, S. Sanghavi, and H. Vikalo. “Structured Low-Rank Matrix Factorization for Haplotype Assembly”. In: *IEEE Journal of Selected Topics in Signal Processing* 10.4 (June 2016), pages 647–657. DOI: 10.1109/JSTSP.2016.2547860.

[83] A. Hashemi, B. Zhu, and H. Vikalo. “Sparse Tensor Decomposition for Haplotype Assembly of Diploids and Polyploids”. In: *BMC Genomics* 19.4 (Mar. 2018). 191. DOI: 10.1186/s12864-018-4551-y.

[84] E. Siragusa, N. Haiminen, R. Finkers, R. Visser, and L. Parida. “Haplotype assembly of autotetraploid potato using integer linear programing”. In: *Bioinformatics* 35.18 (Jan. 2019), pages 3279–3286. DOI: 10.1093/bioinformatics/btz060.

[85] M.-H. Moeinzadeh, J. Yang, E. Muzychenko, G. Gallone, D. Heller, K. Reinert, S. Haas, and M. Vingron. “Ranbow: A fast and accurate method for polyploid haplotype reconstruction”. In: *PLOS Computational Biology* 16.5 (May 2020), pages 1–23. DOI: 10.1371/journal.pcbi.1007843.

[86] S. Mazrouee and W. Wang. “PolyCluster: Minimum Fragment Disagreement Clustering for Polyploid Phasing”. In: *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 17.1 (2020), pages 264–277. DOI: 10.1109/TCBB.2018.2858803.

[87] N. Bansal, A. Blum, and S. Chawla. “Correlation Clustering”. In: *Machine Learning* 56.1 (July 2004), pages 89–113. DOI: 10.1023/B:MACH.0000033116.57574.95.

[88] C. Zahn. “Approximating Symmetric Relations by Equivalence Relations”. In: *Journal of the Society for Industrial & Applied Mathematics* 12 (Dec. 1964), pages 840–847. DOI: 10.1137/0112071.

[89] R. Shamir, R. Sharan, and D. Tsur. “Cluster graph modification problems”. In: *Discrete Applied Mathematics* 144.1 (2004). Discrete Mathematics and Data Mining, pages 173–182. DOI: https://doi.org/10.1016/j.dam.2004.01.007.

[90] E. D. Demaine and N. Immorlica. “Correlation Clustering with Partial Information”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Edited by S. Arora, K. Jansen, J. D. P. Rolim, and A. Sahai. Springer Berlin Heidelberg, 2003, pages 1–13. ISBN: 978-3-540-45198-3. DOI: 10.1007/978-3-540-45198-3_1.

[91] O. Abou Saada, A. Tsouris, C. Eberlein, A. Friedrich, and J. Schacherer. “nPhase: an accurate and contiguous phasing method for polyploids”. In: *Genome Biology* 22.1 (2021), page 126. DOI: 10.1186/s13059-021-02342-x.

[92] R. Shirali Hossein Zade, A. Urhan, A. Assis de Souza, A. Singh, and T. Abeel. “HAT: haplotype assembly tool using short and error-prone long reads”. In: *Bioinformatics* 38.24 (Oct. 2022), pages 5352–5359. DOI: 10.1093/bioinformatics/btac702.

[93] O. A. Saada, A. Friedrich, and J. Schacherer. “Towards accurate, contiguous and complete alignment-based polyploid phasing algorithms”. In: *Genomics* 114.3 (2022). 110369. DOI: 10.1016/j.ygeno.2022.110369.

[94] S. Rahmann, T. Wittkop, J. Baumbach, M. Martin, A. Truss, and S. Böcker. “Exact and Heuristic Algorithms for Weighted Cluster Editing”. In: *Computational systems bioinformatics / Life Sciences Society. Computational Systems Bioinformatics Conference* 6 (Feb. 2007), pages 391–401. DOI: 10.1142/9781860948732_0040.

[95] S. Böcker, S. Briesemeister, and G. W. Klau. “Exact Algorithms for Cluster Editing: Evaluation and Experiments”. In: *Algorithmica* 60.2 (June 2011), pages 316–334. DOI: 10.1007/s00453-009-9339-7.

[96] L. H. N. Lorena, M. G. Quiles, A. C. P. d. L. F. de Carvalho, and L. A. N. Lorena. “Preprocessing Technique for Cluster Editing via Integer Linear Programming”. In: *Intelligent Computing Theories and Application*. Edited by D.-S. Huang, V. Bevilacqua, P. Premaratne, and P. Gupta. Cham: Springer International Publishing, 2018, pages 287–297. ISBN: 978-3-319-95930-6. DOI: 10.1007/978-3-319-95930-6_27.

[97] J. T. Robinson, H. Thorvaldsdóttir, W. Winckler, M. Guttman, E. S. Lander, G. Getz, and J. P. Mesirov. “Integrative genomics viewer”. In: *Nature Biotechnology* 29.1 (Jan. 2011), pages 24–26. DOI: 10.1038/nbt.1754.

[98] Y. Ono, K. Asai, and M. Hamada. “PBSIM: PacBio reads simulatortoward accurate genome assembly”. In: *Bioinformatics* 29.1 (Nov. 2012), pages 119–121. DOI: 10.1093/bioinformatics/bts649.

[99] M. A. Hardigan, E. Crisovan, J. P. Hamilton, J. Kim, P. Laimbeer, C. P. Leisner, N. C. Manrique-Carpintero, L. Newton, G. M. Pham, B. Vaillancourt, X. Yang, Z. Zeng, D. S. Douches, J. Jiang, R. E. Veilleux, and C. R. Buell. "Genome reduction uncovers a large dispensable genome and adaptive role for copy number variation in asexually propagated *Solanum tuberosum*". In: *Plant Cell* 28.2 (Feb. 2016), pages 388–405. DOI: 10.1105/tpc.15.00538.

[100] W. Zhang, B. Jia, and C. Wei. "PaSS: a sequencing simulator for PacBio sequencing". In: *BMC Bioinformatics* 20.1 (June 2019). 352. DOI: 10.1186/s12859-019-2901-7.

[101] C. Yang, J. Chu, R. L. Warren, and I. Birol. "NanoSim: nanopore sequence read simulator based on statistical characterization". In: *Gigascience* 6.4 (Apr. 2017), pages 1–6. DOI: 10.1093/gigascience/gix010.

[102] M. Petek, M. Zagorak,. Ramak, S. Sanders,. Toma, E. Tseng, M. Zouine, A. Coll, and K. Gruden. "Cultivar-specific transcriptome and pan-transcriptome reconstruction of tetraploid potato". In: *Scientific Data* 7.1 (July 2020). 249. DOI: 10.1038/s41597-020-00581-4.

[103] A. V. Zimin, D. Puiu, M.-C. Luo, T. Zhu, S. Koren, G. Marçais, J. A. Yorke, J. Dvoák, and S. L. Salzberg. "Hybrid assembly of the large and highly repetitive genome of *Aegilops tauschii*, a progenitor of bread wheat, with the MaSuRCA mega-reads algorithm". In: *Genome Research* 27.5 (May 2017), pages 787–792. DOI: 10.1101/gr.213405.116.

[104] P. P. Edger, T. J. Poorten, R. VanBuren, M. A. Hardigan, M. Colle, M. R. McKain, R. D. Smith, S. J. Teresi, A. D. L. Nelson, C. M. Wai, E. I. Alger, K. A. Bird, A. E. Yocca, N. Pumplin, S. Ou, G. Ben-Zvi, A. Brodt, K. Baruch, T. Swale, L. Shiue, C. B. Acharya, G. S. Cole, J. P. Mower, K. L. Childs, N. Jiang, E. Lyons, M. Freeling, J. R. Puzey, and S. J. Knapp. "Origin and evolution of the octoploid strawberry genome". In: *Nature Genetics* 51.3 (Mar. 2019), pages 541–547. DOI: 10.1038/s41588-019-0356-4.

[105] X. Jin, H. Du, C. Zhu, H. Wan, F. Liu, J. Ruan, J. P. Mower, and A. Zhu. "Haplotype-resolved genomes of wild octoploid progenitors illuminate genomic diversifications from wild relatives to cultivated strawberry". In: *Nature Plants* 9.8 (Aug. 2023), pages 1252–1266. DOI: 10.1038/s41477-023-01473-2.

[106] D. J. Bertioli, S. B. Cannon, L. Froenicke, G. Huang, A. D. Farmer, E. K. S. Cannon, X. Liu, D. Gao, J. Clevenger, S. Dash, L. Ren, M. C. Moretzsohn, K. Shirasawa, W. Huang, B. Vidigal, B. Abernathy, Y. Chu, C. E. Niederhuth, P. Umale, A. C. G. Araújo, A. Kozik, K. Do Kim, M. D. Burow, R. K. Varshney, X. Wang, X. Zhang, N. Barkley, P. M. Guimarães, S. Isobe, B. Guo, B. Liao, H. T. Stalker, R. J. Schmitz, B. E. Scheffler, S. C. M. Leal-Bertioli, X. Xun, S. A. Jackson, R. Michelmore, and P. Ozias-Akins. "The genome sequences of Arachis duranensis and Arachis ipaensis, the diploid ancestors of cultivated peanut". In: *Nature Genetics* 48.4 (Apr. 2016), pages 438–446. DOI: 10.1038/ng.3517.

[107] X. Jiang, D. Li, H. Du, P. Wang, L. Guo, G. Zhu, and C. Zhang. “Genomic features of meiotic crossovers in diploid potato”. In: *Horticulture Research* 10.6 (Apr. 2023). DOI: 10.1093/hr/uhad079.

[108] C. R. Clot, X. Wang, J. Koopman, A. T. Navarro, J. Bucher, R. G. F. Visser, R. Finkers, and H. J. van Eck. “High-Density Linkage Map Constructed from a Skim Sequenced Diploid Potato Population Reveals Transmission Distortion and QTLs for Tuber Yield and Pollen Shed”. In: *Potato Research* 67.1 (Mar. 2024), pages 139–163. DOI: 10.1007/s11540-023-09627-7.

[109] E. M. Arkin and E. B. Silverberg. “Scheduling jobs with fixed start and end times”. In: *Discrete Applied Mathematics* 18.1 (1987), pages 1–8. DOI: 10.1016/0166-218X(87)90037-0.

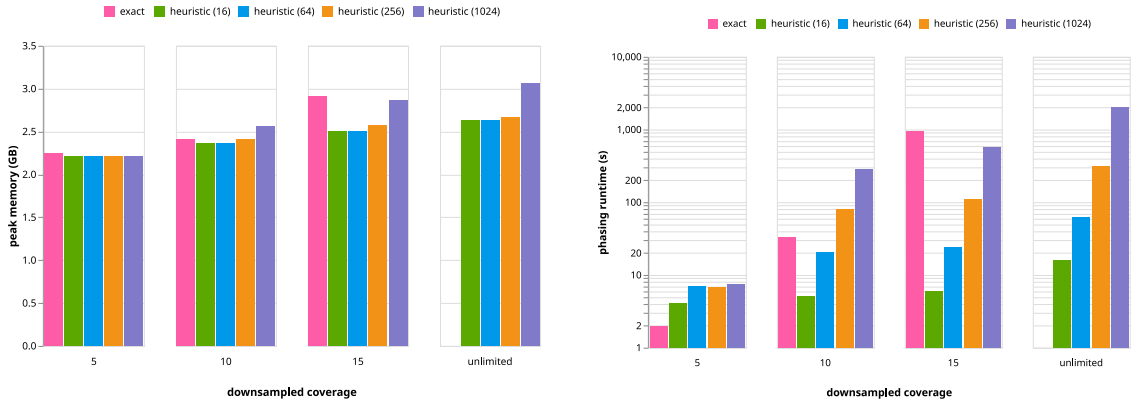
[110] S. Nurk et al. “The complete sequence of a human genome”. In: *Science* 376.6588 (2022), pages 44–53. DOI: 10.1126/science.abj6987.

[111] T. Wang, L. Antonacci-Fulton, K. Howe, H. A. Lawson, J. K. Lucas, A. M. Phillippy, A. B. Popejoy, M. Asri, C. Carson, M. J. P. Chaisson, X. Chang, R. Cook-Deegan, A. L. Felsenfeld, R. S. Fulton, E. P. Garrison, N. A. Garrison, T. A. Graves-Lindsay, H. Ji, E. E. Kenny, B. A. Koenig, D. Li, T. Marschall, J. F. McMichael, A. M. Novak, D. Purushotham, V. A. Schneider, B. I. Schultz, M. W. Smith, H. J. Sofia, T. Weissman, P. Flicek, H. Li, K. H. Miga, B. Paten, E. D. Jarvis, I. M. Hall, E. E. Eichler, D. Haussler, and the Human Pangenome Reference Consortium. “The Human Pangenome Project: a global resource to map genomic diversity”. In: *Nature* 604.7906 (Apr. 2022), pages 437–446. DOI: 10.1038/s41586-022-04601-8.

Appendix A

Additional results for the (Ped)MEC heuristic

A.1 Additional benchmarks for the single-sample datasets



(a) Peak memory consumption

(b) Runtime for phasing algorithm only

Figure A.1: **Memory consumption and runtime of MEC heuristic (HiFi).** This figure extends Figure 2.8 by the peak memory consumption and phasing runtime.

A.2 Additional benchmarks for trio-phasing datasets

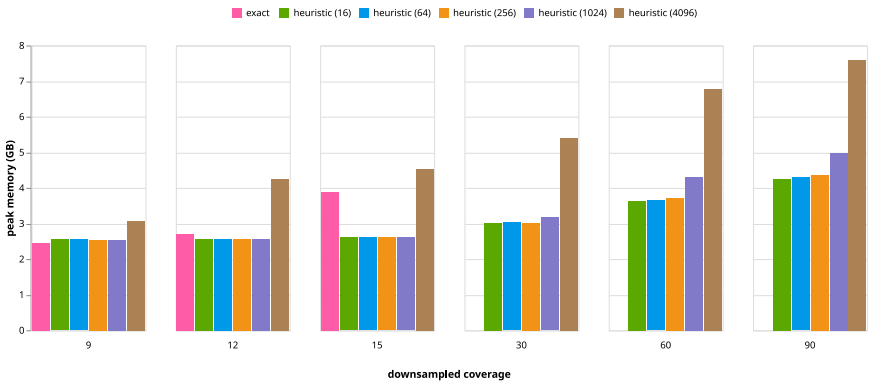
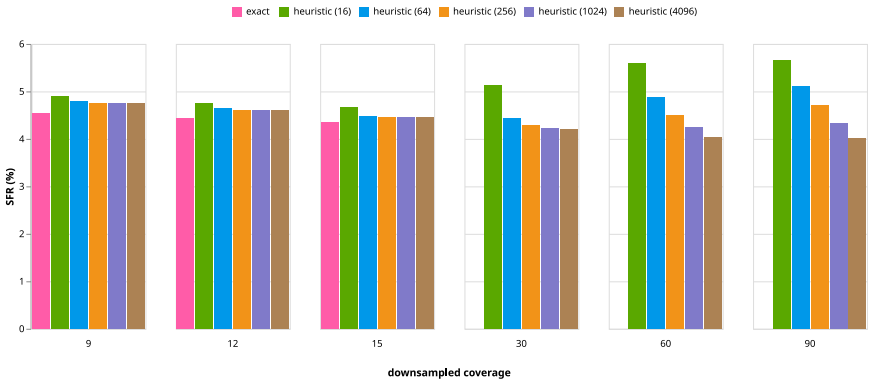
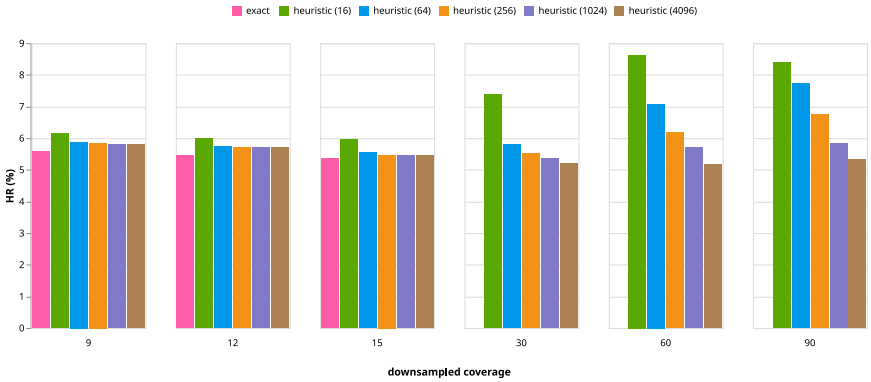


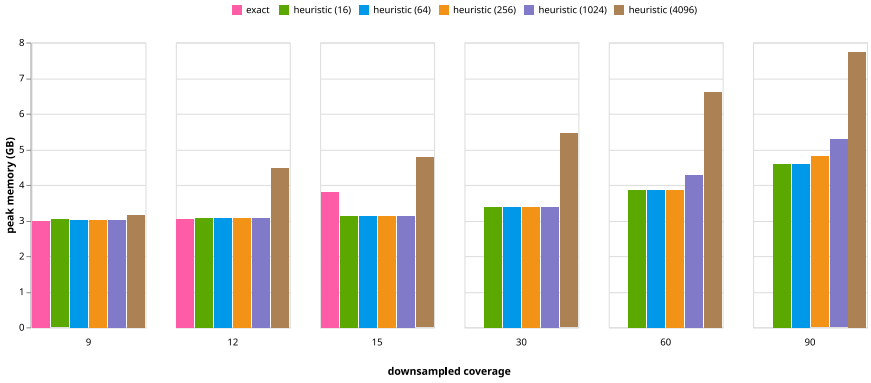
Figure A.2: **Memory consumption for PedMEC evaluation.** This figure extends Figure 2.9 by the peak memory consumption.



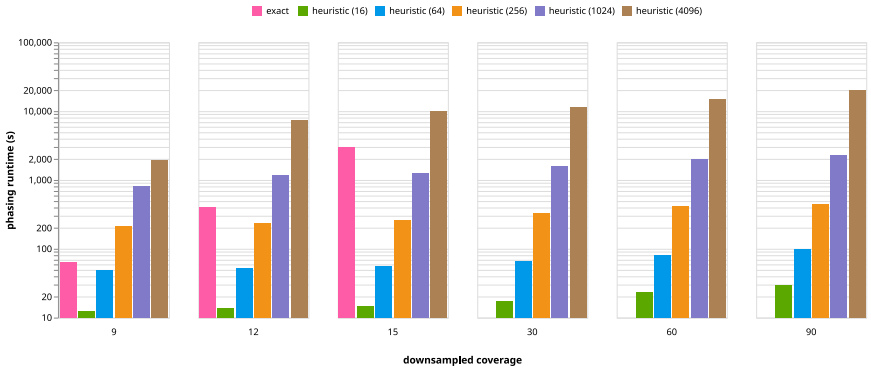
(a) Switch flip rate



(b) Hamming rate



(c) Peak memory consumption



(d) Runtime for phasing algorithm only

Figure A.3: **Evaluation of PedMEC heuristic (HiFi).** These plots contain the same results as described in Figure 2.7 but for the HiFi data instead of the PacBio data. All algorithms used the PedMEC model and reads from all three samples.

Appendix B

Additional results for WHATSHAP POLYPHASE

B.1 Benchmarks on artificial polyploid human data

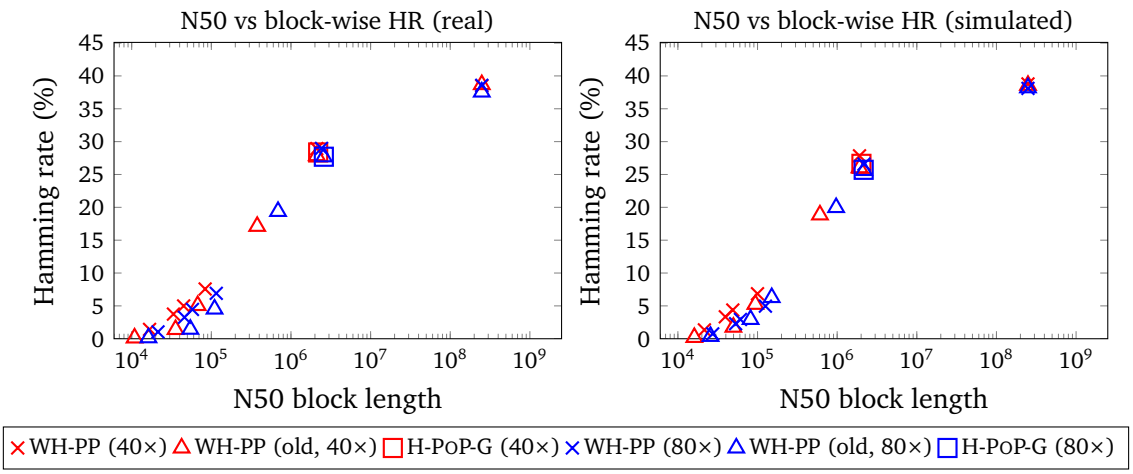


Figure B.1: **N50 block lengths vs block-wise Hamming rates.** N50 block lengths and the respective block-wise Hamming rates for different block cut strategies of WHATSHAP POLYPHASE on the real tetraploid read dataset (top) and the simulated tetraploid dataset (bottom) with 40× and 80× coverage.

coverage	method	SFR (%)	SER (%)	HR (%)	WGR (%)	N50 (bp)
40×	WH-PP	3.90	5.32	6.82	0	16641
	WH-PP$_{\text{OLD}}$	1.41	1.76	2.35	0	18635
	WH-PP*	4.90	6.55	29.26	0	2435940
	WH-PP*$_{\text{OLD}}$	2.60	3.18	28.49	0	2421639
	H-PoP-G	3.96	5.69	28.08	0	2587104
	FLOPP	8.36	3.64	36.75	16.43	248889649
80×	WH-PP	3.19	4.37	6.25	0	23471
	WH-PP$_{\text{OLD}}$	0.81	0.95	1.58	0	25439
	WH-PP*	3.94	5.28	27.98	0	2587104
	WH-PP*$_{\text{OLD}}$	1.78	2.08	27.09	0	2587104
	H-PoP-G	3.08	4.38	27.46	0	2599743
	FLOPP	10.21	2.72	38.37	29.71	248889649

Table B.1: Comparison of WHATSHAP POLYPHASE, H-PoP-G and FLOPP on pentaploid simulated datasets. Performances are based on switch flip rate (SFR), switch error rate (SER), block-wise Hamming rate (HR), wrong genotype rate (WGR) and N50 block size.

coverage	method	SFR (%)	SER (%)	HR (%)	WGR (%)	N50 (bp)
40×	WH-PP	1.98	2.43	3.60	0	8758
	WH-PP$_{\text{OLD}}$	0.84	1.00	1.67	0	16028
	WH-PP^{d}	6.95	1.03	7.13	32.74	10349
	WH-PP*	3.42	4.10	27.79	0	4265325
	WH-PP*$_{\text{OLD}}$	1.96	2.27	27.30	0	4265325
	WH-PP*^{d}	7.96	1.66	30.94	32.81	4265325
80×	WH-PP	1.27	1.52	2.83	0	15162
	WH-PP$_{\text{OLD}}$	0.48	0.53	1.14	0	25429
	WH-PP^{d}	4.50	0.67	5.01	21.54	16240
	WH-PP*	2.30	2.70	27.94	0	6412111
	WH-PP*$_{\text{OLD}}$	1.27	1.41	25.68	0	6412111
	WH-PP*^{d}	5.37	1.33	30.39	21.73	6412111

(a) real hexaploid read data

coverage	method	SFR (%)	SER (%)	HR (%)	WGR (%)	N50 (bp)
40×	WH-PP	2.07	2.64	3.45	0	8947
	WH-PP (old)	0.88	1.06	1.83	0	16882
	WH-PP ^{d}	7.54	1.02	7.44	35.14	11005
	WH-PP *	3.43	4.24	27.52	0	3754960
	WH-PP (old)	1.99	2.31	26.96	0	3754960
	WH-PP *^{d}	8.50	1.55	30.85	34.91	3754960
80×	WH-PP	1.03	1.25	2.12	0	14656
	WH-PP$_{\text{OLD}}$	0.43	0.48	0.96	0	26251
	WH-PP^{d}	4.16	0.57	4.36	20.11	15679
	WH-PP*	2.04	2.40	26.55	0	4490129
	WH-PP*$_{\text{OLD}}$	1.23	1.36	26.08	0	4607645
	WH-PP*^{d}	5.04	1.23	28.58	20.33	4490129

(b) simulated hexaploid read data

Table B.2: Comparison of WHATSHAP POLYPHASE and FLOPP, with and without trusting genotypes.

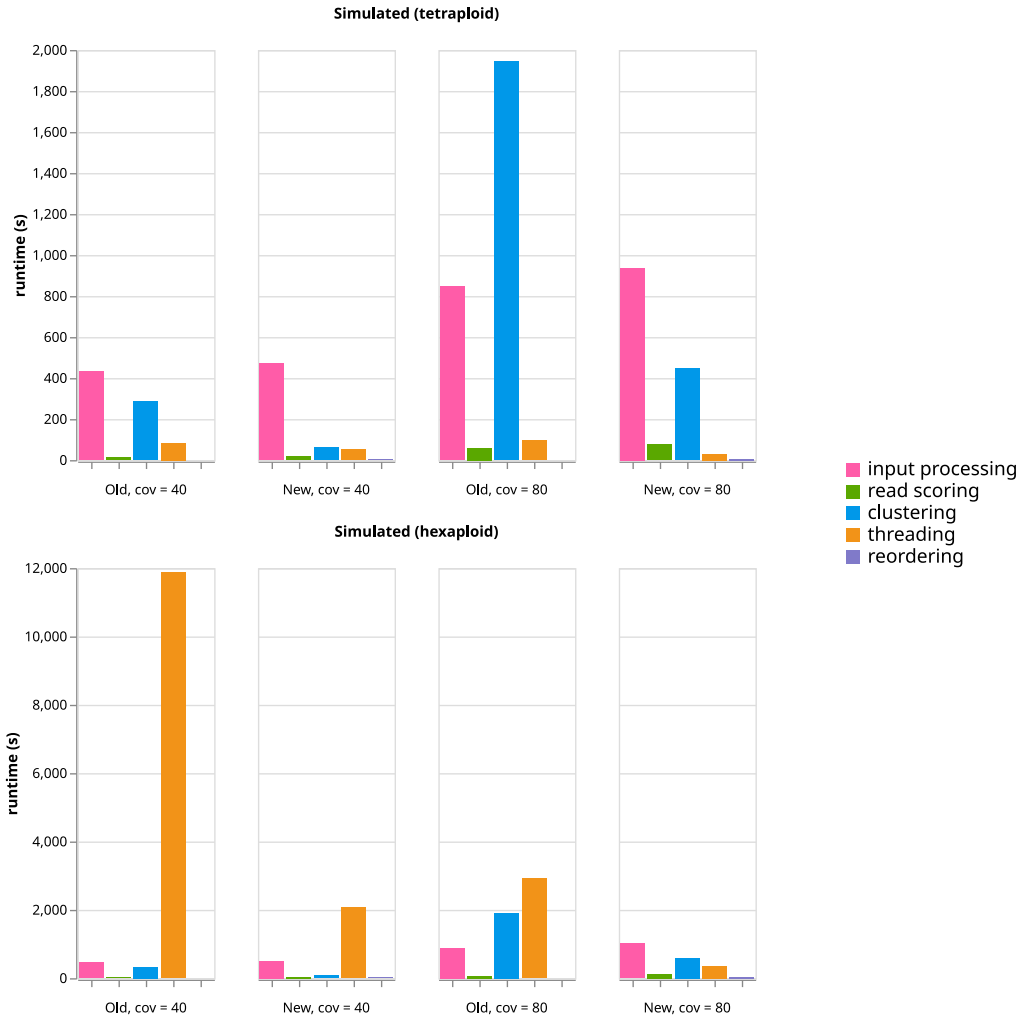


Figure B.2: **Runtime of different WhatsHap Polyphase stages.** For the simulated datasets, this plot shows the runtime of the different stages of both old and new versions of WhatsHap Polyphase. The left and right groups of bar plots refer to the tetraploid and hexaploid datasets, respectively.

The comparison between collapsing and non-collapsing regions was done by Rebecca Serra Mari [14]. The numbers and the table description are taken from the paper.

coverage	method	collapsing regions	non-collapsing regions	total
40×	WH-PP$_{\text{OLD}}$	0.29	0.69	0.60
	H-PoP-G	2.02	2.16	2.02
	SER($\frac{\text{H-PoP-G}}{\text{WH-PP}_{\text{OLD}}}$)	6.97	3.13	3.37
80×	WH-PP$_{\text{OLD}}$	0.14	0.46	0.35
	H-PoP-G	1.05	1.30	1.24
	SER($\frac{\text{H-PoP-G}}{\text{WH-PP}_{\text{OLD}}}$)	7.50	2.83	3.54

(a) Real read data (tetraploid)

coverage	method	collapsing regions	non-collapsing regions	total
40×	WH-PP$_{\text{OLD}}$	0.18	0.45	0.43
	H-PoP-G	2.01	1.63	1.68
	SER($\frac{\text{H-PoP-G}}{\text{WH-PP}_{\text{OLD}}}$)	11.17	3.62	3.91
80×	WH-PP$_{\text{OLD}}$	0.08	0.37	0.32
	H-PoP-G	0.94	0.98	0.99
	SER($\frac{\text{H-PoP-G}}{\text{WH-PP}_{\text{OLD}}}$)	11.75	2.65	3.09

(b) Simulated read data (tetraploid)

Table B.3: Comparison between the resulting switch error rates of the original version of WHATSHAP POLYPHASE (WH-PP$_{\text{OLD}}$) and H-PoP-G on collapsing regions over at least 50 variants as compared to non-collapsing regions and the average throughout the genome. Results (switch error rates in %) are presented for Chromosome 1 of the real (a) and simulated (b) tetraploid dataset on both 40× and 80× coverage. The third row marks the quotient between the switch error rate of H-PoP-G and that of WHATSHAP POLYPHASE to highlight by which magnitude the results differ.

B.2 Benchmarks on simulated Solanum tuberosum data

method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants	N50 (bp)
WH-PP*	0.13	0.08	0.10	0	67461	3019508
WH-PP*^{d}	0.34	0.01	0.72	1.29	67044	3018008
WH-PP$^{*}_{\text{OLD}}$	0.36	0.51	36.75	0	67390	3018244
H-PoP-G	0.46	0.79	0.47	0	67721	3019149
FLOPP	1.14	0.01	1.14	4.23	64995	3016505

(a) simulated tetraploid PacBio reads

method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants	N50 (bp)
WH-PP*	0.19	0.30	23.78	0	66407	3019618
WH-PP*^{d}	0.55	0.01	12.60	2.56	65850	3019084
WH-PP$^{*}_{\text{OLD}}$	0.46	0.61	34.03	0	66313	3019084
H-PoP-G	0.37	0.65	0.38	0	66609	3018966
FLOPP	1.28	0.01	1.28	5.54	64244	3014872

(b) simulated pentaploid PacBio reads

Table B.4: Comparison between WHATSHAP POLYPHASE, H-PoP-G and FLOPP on simulated Solanum tuberosum data.

B.3 Benchmarks on Altus cultivar data

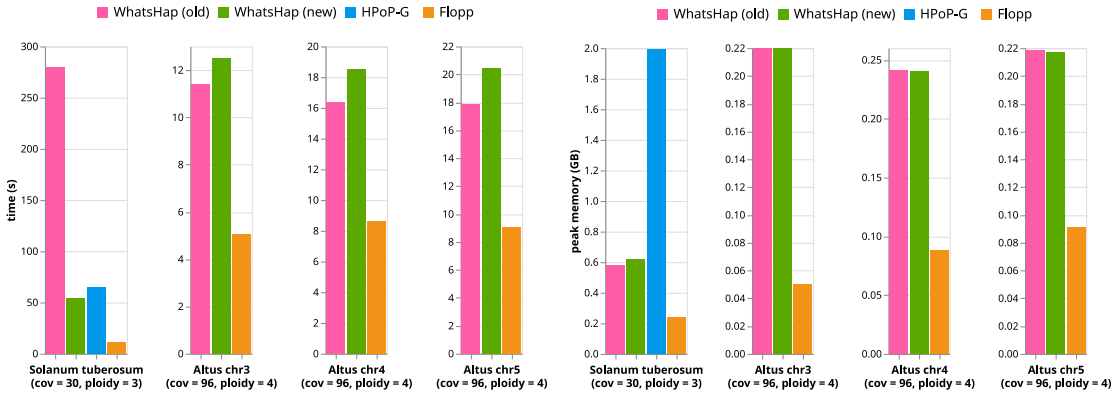
method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants
WH-PP*	7.69	0.42	21.02	28.01	5364
WH-PP$^*_{\text{OLD}}$	9.36	1.94	36.80	28.71	5475
WH-PP^{*d}	2.53	0.10	8.16	9.35	5237
FLOPP	3.23	0.09	20.16	11.91	4886

(a) region ch03:60,269,000-60,504,000

method	SFR (%)	SER (%)	HR (%)	WGR (%)	variants
WH-PP*	7.67	1.20	16.90	24.83	7468
WH-PP$^*_{\text{OLD}}$	9.05	2.24	38.08	25.34	7515
WH-PP^{*d}	3.11	0.08	21.99	11.73	7140
FLOPP	0.42	0.04	5.98	1.27	7487

(b) region ch05:56,711,000-57,066,000

Table B.5: Additional benchmarks for Hifi reads from Altus cultivar.



(a) Runtime in seconds for different datasets.

(b) Peak memory consumption in GiB for the same datasets.

Figure B.3: **Runtime and memory consumption.** Each bar plot represents one dataset, which is stated below the x-axis. The first dataset corresponds to the simulated Solanum tuberosum data from Section 3.5.2, while the others belong to the three test regions on the Altus sample (Section 3.5.3). The left plot (a) shows the runtime in seconds for each algorithm. The right plot (b) shows the peak memory consumption in GiB. WHATSHAP POLYPHASE was run with block cut sensitivity `-B 1`.

Appendix C

Additional results for WHATSHAP POLYPHASE-GENETIC

C.1 Detailed benchmarks on selected regions

variant types	phased	SER (%)	WGR (%)	HR (%)
only simplex-nulliplex (no retyping)	4091	2.94	10.83	7.06
only simplex-nulliplex	3934	0.97	3.46	2.89
(50% sampled)	3581	0.86	3.27	2.55
(25% sampled)	3164	0.50	6.89	4.2
additional simplex-simplex	3927	0.95	3.26	2.63
additional duplex-nulliplex	5943	3.25	20.46	8.88

Table C.1: **Error metrics for WH-PPG on chromosome 3 region.** Region coordinates are ch03:60,269,000-60,504,000 with 9549 bi-allelic and 10286 total variants. The exclusive simplex-nulliplex-mode has been repeated with 50% and 25% of parental coverage for genotype calling and without retyping. The variant type additions are cumulative, i.e. the last row shows the results for all three variant types.

variant types	phased	SER (%)	WGR (%)	HR (%)
only simplex-nulliplex (no retyping)	3170	0.43	0.82	1.77
only simplex-nulliplex	3127	0.26	1.37	0.63
(50% sampled)	2948	0.47	1.42	0.81
(25% sampled)	2562	0.51	1.83	1.08
additional simplex-simplex	4289	1.37	2.19	1.63
additional duplex-nulliplex	4508	1.57	3.02	2.02

Table C.2: **Error metrics for WH-PPG on chromosome 4 region.** Region coordinates are ch04:71,586,000-71,947,000 with 12378 bi-allelic and 14500 total variants. Columns and rows follow the same scheme as Table C.1.

variant types	phased	SER (%)	WGR (%)	HR (%)
only simplex-nulliplex (no retyping)	4552	1.29	1.25	1.47
only simplex-nulliplex	5096	0.82	0.90	0.92
(50% sampled)	4634	0.47	0.73	0.56
(25% sampled)	4075	0.52	0.69	0.60
additional simplex-simplex	5332	0.73	1.41	0.95
additional duplex-nulliplex	6350	0.99	9.78	3.33

Table C.3: **Error metrics for WH-PPG on chromosome 5 region.** Region coordinates are ch05:56,711,000-57,066,000 with 13030 bi-allelic and 15810 total variants. Columns and rows follow the same scheme as Table C.1.

Appendix D

Code and data availability

The main repository for all workflow pipelines and instructions can be found at `https://github.com/schrins/dissertation-pipelines`. It will be updated in case of bug fixes or improvements. The initial release of this repository is also archived on Zenodo under `https://zenodo.org/doi/10.5281/zenodo.11266453`.

Data availability is described in the above-mentioned repository. In addition, important intermediate files created during the experiments are archived on Zenodo at `https://doi.org/10.5281/zenodo.11264527`.

The main tool WhatsHap is available at `https://github.com/whatshap/whatshap`. The custom version that was used for some of the experiments refers to commit `77133e6` in branch `sven-thesis`. It can be accessed through `https://github.com/whatshap/whatshap/commit/77133e665616346f66606c6dbdae407c97af6c29`.

Appendix E

Published articles contributing to this thesis

E.1 Haplotype Threading: Accurate Polyploid Phasing from Long Reads

E.1.1 Authors

Sven D. Schrinner*, Rebecca Serra Mari*, Jana Ebler*, Mikko Rautiainen, Lancelot Seillier, Julia J. Reimer, Björn Usadel, Tobias Marschall and Gunnar W. Klau

**shared first authors*

E.1.2 Contributions

The following quotes the author contributions as stated in the published article [14]:

> SDS, RSM, JE, GWK, and TM developed the algorithmic concepts and designed the study. RSM designed the haplotype threading algorithm and implemented a prototype. SDS designed and implemented the cluster editing algorithm, designed the block cut strategies, and optimized the threading implementation. JE performed the evaluation and analyzed the potato dataset. MR ran the error correction on the potato reads. LS, JJR, and BU performed potato sequencing, and BU helped with the interpretation of phasing results. SDS, RSM, and JE integrated all software components into WhatsHap and tested the workflow. SDS, RSM, JE, GWK, and TM wrote the paper. All authors read and approved the final manuscript.

E.1.3 License and copyright

The "Rights and permissions" section on the publication's web address (https://doi.org/10.1186/s13059-020-02158-1) states the following:

E.2 Genetic polyploid phasing from low-depth progeny samples

E.2.1 Authors

Sven Schrinner, Rebecca Serra Mari, Richard Finkers, Paul Arens, Björn Usadel, Tobias Marschall, Gunnar W. Klau

E.2.2 Contributions

The following quotes the author contributions as stated in the published article [16]:

> Conceptualization, S.S., R.S.M., R.F., B.U., T.M. and G.W.K.; Methodology S.S., R.S.M., R.F., T.M. and G.W.K.; Software S.S.; Investigation S.S., R.S.M., R.F., B.U., T.M. and G.W.K.; Resources R.F., P.A., and B.U.; Writing Original Draft S.S. and G.W.K.; Writing Review & Editing S.S., R.S.M., R.F., P.A., B.U., T.M. and G.W.K.;

E.2.3 License and copyright

https://doi.org/10.1016/j.isci.2022.104461

E.3 Haplotype-resolved assembly of a tetraploid potato genome using long reads and low-depth offspring data

E.3.1 Authors

Rebecca Serra Mari, Sven Schrinner, Richard Finkers, Freya Maria Rosemarie Ziegler, Paul Arens, Maximilian H.-W. Schmidt, Björn Usadel, Gunnar W. Klau & Tobias Marschall

E.3.2 Contributions

The following quotes the author contributions as stated in the published article [52]:

> R.S.M. and T.M. designed the assembly method, with input from S.S., R.F., B.U., and G.W.K. R.S.M. implemented the assembly method, performed the assemblies, and created the figures. R.S.M., S.S., R.F., G.W.K., B.U., and T.M. discussed and interpreted results. R.S.M., R.F., B.U., and T.M. wrote the manuscript, with input from S.S. and G.W.K. P.A. provided offspring sequencing data. M.HW.S. produced HiFi data and ran an initial assembly. F.M.R.Z. produced ONT data and wrote text describing data production. All authors read and approved the final manuscript.

E.3.3 License and copyright

The "Rights and permissions" section on the publication's web address (https://doi.org/10.1186/s13059-023-03160-z) states the following:

List of Tables

List of Figures

List of variables and symbols

Symbol	Chapter	Meaning or usage
$\mathcal{A}$	1.4	Input allele matrix
$a_{\max}$	1.4	Highest number of alternative alleles
A_j	2.2.1	Set of read indices ($\subseteq \{1, \ldots, n\}$) that is active for column/variant j
A^+, A^-	3.4.1	Set of excess alleles, underrepresented alleles
A	3.4.1	Union of A^+ and A^-
A_{place}	3.4.1	Multiset of alleles to redistribute
$a_{b,t,h}$	3.4.3	Score of placing thread t on haplotype h for block b according to pre-phasing
$\mathscr{B}(A)$	2.2.1	The set of all bipartitions over set A
$\mathscr{B}(A \mid B)$	2.2.1	The set of all bipartitions over set A that extend bipartition B
$\mathcal{B}$, $\mathcal{B}_b$	3.4.3	Set of breakpoints, b-th breakpoint
$\mathcal{C}$	3.2.3	Set of all read clusters
C_j	3.2.3	The j-th read cluster in $\mathcal{C}$
c_j	3.4.1	j-th cluster index in some cluster tuple $(c_1, \ldots, c_p) \in \{1, \ldots, \lvert\mathcal{C}\rvert\}^p$
cost_{cov}	3.3	Total coverage penalties for a threading
$\text{cost}_{\text{switch}}$	3.3	Total switch penalties for a threading
$\text{cov}(C, i)$	3.3	Absolute coverage of v_i by all reads in cluster C
$\text{cov}_{\text{r}}(C, i)$	3.3	Relative coverage of v_i by all reads in cluster C
$D, D(B, j)$	2.2.1	DP table for exact MEC solver, entry for column j and bipartition B
$D', D'(B, j)$	2.2.1	DP table for heuristic MEC solver, entry for column j and bipartition B
$D_{a,j}()$	3.4.1	Number of occurrence (allele depth) for allele a in cluster $C_j \in \mathcal{C}$
$D_{C_j,i}(a)$	3.3	Allele depth of allele a for cluster C_j at variant v_i
$e(r)$	1.4	Last covered position by read r
$e(C)$	3.3	Last covered position by any read in cluster C
$f_a(G)$	1.4	Absolute frequency of allele a in genotype G
F, F_i	2.1.2	Set of computed flips for the MEC model (optionally for the i-th sample)

G_i	1.4	Input genotype for variant v_i
G'_i	3.4.1	Induced genotypes by some intermediate haplotypes for variant v_i
$\mathcal{H}, H_i$	1.4	Set of true haplotype, i-th element
$\tilde{\mathcal{H}}, \tilde{H}_i$	1.4	Set of predicted haplotype, i-th element
$\overline{H}, \overline{H}_i$	1.4	Set of pre-phased haplotype, i-th element
H_i^s	2.1.3	i-th haplotype of sample s
I	3.4.1	Haplotype indices containing an excess allele
$\mathcal{I}$	2.1.3	Family of related samples to phase
$\text{icp}(uv), \text{icf}(uv)$	3.2	Induced cost permanent/forbidden for edge uv
$L_{C_j,i}$	3.3	Consensus list of cluster C_j for variant v_i
l	3.4.3	Number of blocks for reordering
$l_{b,i}$	3.4.3	Likelihood for i-th rearrangements of threads over breakpoint b
$L_=$, $L_\neq$	4.2.1	Probability to inherit certain alleles for simplex-nulliplex variants
$L_=^S$, $L_\neq^S$ / $L_=^D$, $L_\neq^D$	4.2.1	Same as $L_=$, $L_\neq$ for simplex-simplex/duplex-nulliplex variants
m	1.4	Total number of variants to phase
$\text{mult}(i,j)$	3.3	Selected (or induced) multiplicity of cluster C_j at v_i
$\text{mult}_{\text{exp}}(i,j)$	3.3	Expected multiplicity of cluster C_j at variant i
n	1.4	Total number of reads to be used
N_j	2.2.1	Set of read indices ($\subseteq \{1,\ldots,n\}$) that newly start at variant j
p	1.4	Global ploidy
p_{affine}	3.3	Affine penalty for cluster switches during haplotype threading
p_{cov}	3.3	Penalty for deviating coverage during haplotype threading
p_{switch}	3.3	Penalty for cluster switches during haplotype threading
$\mathcal{R}$, $\mathcal{R}_i$	1.4	Set of input reads, i-th read
$s(r)$	1.4	First covered position by read r
$s(C)$	3.3	First covered position by any read in cluster C
$\text{span}(v_i)$	3.3	Set of clusters spanning variant v_i
$S, S(i,j)$	3.3.3	DP table for threading, entry for variant j and i cluster tuple
$\mathcal{T}$	3.3	Threading, tuple of all threads
T_b^{aff}	3.4.3	Affected threads ($\subseteq \{1,\ldots,p\}$) for b-th breakpoint
$T_i, T_{(i)}$	3.3	For threading $\mathcal{T}$ the i-th thread and i cluster tuple, respectively
$t_{i,j}$	3.3	For variant v_i, the j-th inspected candidate cluster tuple
$\mathcal{T}$	2.1.3	Set of trios (mother, father and child) for family $\mathcal{I}$
$t_{i_p \to i_c}$	2.1.3	Transmission vector for parent i_p to child i_c
v_i	1.4	the i-th variant of the region ot phase
$\mathcal{V}$	1.4	Set of input variants
$\mathcal{W}$	2.1.2	Flip cost matrix with the same dimension of allele matrix $\mathcal{A}$
W	4.2.2	Window size for variant scoring in WHATSHAP POLYPHASE-GENETIC
$\mathcal{X}$	2.1.3	Vector of position-wise recombination cost
$X(B,j), X(B,j)_q$	2.2.1	Allele balance vector (for partition $q \in \{1,2\}$)
$\mathcal{Y}$	2.2.2	Vector of position-wise mutation cost