Computational Haplotyping: theory and practice

Shilpa Garg

Dissertation submitted towards the degree Doctor of Engineering
of the Faculty of Mathematics and Computer Science
of Saarland University

Saarbrücken
2018

Colloquium

Date	:	June 21, 2018
Place	:	Saarbrücken
Dean	:	Prof. Sebestian Hack

Examination Board

Chairman	:	Prof. Hans-Peter Lenhof
Reviewer	:	Prof. Tobias Marschall
Reviewer	:	Prof. Volkhard Helms
Scentific Assitant	:	Dr. Christina Backes

Abstract

Genomics has paved a new way to comprehend life and its evolution, and also to investigate causes of diseases and their treatment. One of the important problems in genomic analyses is haplotype assembly. Constructing complete and accurate haplotypes plays an essential role in understanding population genetics and how species evolve. In this thesis, we focus on computational approaches to haplotype assembly from third generation sequencing technologies. This involves huge amounts of sequencing data, and such data contain errors due to the single molecule sequencing protocols employed. Taking advantage of combinatorial formulations helps to correct for these errors to solve the haplotyping problem. Various computational techniques such as dynamic programming, parameterized algorithms, and graph algorithms are used to solve this problem.

This thesis presents several contributions concerning the area of haplotyping. First, a novel algorithm based on dynamic programming is proposed to provide approximation guarantees for phasing a single individual. Second, an integrative approach is introduced to combining multiple sequencing datasets to generating complete and accurate haplotypes. The effectiveness of this integrative approach is demonstrated on a real human genome. Third, we provide a novel efficient approach to phasing pedigrees and demonstrate its advantages in comparison to phasing a single individual. Fourth, we present a generalized graph-based framework for performing haplotype-aware de novo assembly. Specifically, this generalized framework consists of a hybrid pipeline for generating accurate and complete haplotypes from data stemming from multiple sequencing technologies, one that provides accurate reads and other that provides long reads.

Kurzfassung

Die Genomik hat neue Wege eröffnet, die es ermöglichen, die Evolution lebendiger Organismen zu verstehen, sowie die Ursachen zahlreicher Krankheiten zu erforschen und neue Therapien zu entwickeln. Ein wichtiges Problem ist die Assemblierung der Haplotypen eines Individuums. Diese Rekonstruktion von Haplotypen spielt eine zentrale Rolle für das Verständnis der Populationsgenetik und der Evolution einer Spezies. In der vorliegenden Arbeit werden Algorithmen zur Assemblierung von Haplotypen vorgestellt, die auf Sequenzierdaten der dritten Generation basieren. Dies erfordert große Mengen an Daten, welche wiederum Fehler enthalten, die die zugrunde liegenden Sequenzierprotokolle hervorbringen. Durch kombinatorische Formulierungen des Problems ist die Rekonstruktion von Haplotypen dennoch möglich, da Fehler erfolgreich korrigiert werden können. Verschiedene informatische Methoden, wie dynamische Programmierung, parametrisierte Algorithmen und Graph Algorithmen können verwendet werden, um dieses Problem zu lösen.

Die vorliegende Arbeit stellt mehrere Lösungsansätze für die Rekonstruktion von Haplotypen vor. Als erstes wird ein neuartiger Algorithmus vorgestellt, der basierend auf dem Prinzip der dynamischen Programmierung Approximationsgarantien für das Haplotyping eines einzelnen Individuums liefert. Als zweites wird ein integrativer Ansatz präsentiert, um mehrere Sequenzierdatensätze zu kombinieren und somit akkurate Haplotypen zu generieren. Die Effektivität dieser Methode wird auf einem echten, menschlichen Datensatz demonstriert. Als drittes wird ein neuer, effizienter Algorithmus beschrieben, um Haplotypen verwandter Individuen simultan zu konstruieren und die Vorteile gegenüber der Betrachtung einzelner Individuen aufgezeigt. Als viertes präsentieren wir eine Graph-basierte Methode um mittels Haplotypinformation de-novo Assemblierung durchzuführen. Dieser Methode kombiniert Daten stammend von verschiedenen Sequenziertechnologien, welche entweder genaue oder aber lange Sequenzierreads liefern.

Acknowledgements

I would like to express my deep gratitude to all the people who supported me during this work.

First and foremost, I am grateful to my supervisor Dr Tobias Marschall who introduced me to this haplotyping problem. He helped me to develop problem solving skills for solving a research problem and provided me his able guidance in completing this thesis work. Thanks a million to Prof Volkhard Helms for his consent to give time to review this thesis at a short notice. I would like to offer special thanks to Prof Thomas Lengauer for providing me his valuable remarks on an earlier version of some of the chapters of this thesis. He provided me financial support and guided me about ways to pursue my passion. I am thankful to Prof George Church for his valuable comments on an earlier version of this thesis.

I am deeply grateful to Prof Richard Durbin whose lab I visited for my research internship. His enthusiasm for genomics and guidance on future career prospects largely motivated me. Dr Tobias Mömke has been great source of inspiration and advice for me. I am continually amazed by his problem solving ability, which provided me motivation to solve ambitious problems.

I am immensely thankful to all collaborators and coauthors for their support and assistance. It was great working with David Porubsky and Ashley Sanders, and learning the magic of Strand-Seq technology. Thanks to Mikko Rautiainen and vgteam for great discussions on the graph world of genomics.

I appreciate the discussions with my office-mate Ali Ghaffaari on violin and classical music. Special thanks to Jana for always providing a friendly support and guidance. I would like to extend my thanks to my group mates, who organized different events at weekends. The events were enjoyable experiences.

Jana, Fabio, Lara, Prabhav and Adam proofread a few chapters of this thesis and I greatly appreciate their valuable remarks and feedback.

I appreciate the travel financial support from Michelle Carnell, technical help from Achim, George and MPII-IST and other administrative help from Susanne and Ruth.

Finally, I would like to express the gratitude to my family and friends, who greatly helped me in this journey.

Contents

Chapter **1**

Introduction

Genetics and Genomics study the phenomenon of *life* at its most basic level and are like wise fascinating and important. An important field in genetics is haplotyping. Haplotyping is the process of determining the sequences of both copies of homologous chromosomes, which are inherited from each parent in diploid organisms. Haplotyping has applications in different fields such as evolutionary studies, clinical diagnosis, precision medicine, and biotechnology. Third generation sequencing technologies allow for reading fragments (in the order of magnitude of tens of kilo-bases) of the genome sequence, and thus the reconstruction of haplotypes is possible, in principle. Unfortunately, sequencing is prone to errors and the use of advanced algorithms and models is essential to correct for errors, in order to reconstruct accurate haplotypes. However, the process of correcting these errors poses various computational challenges. In this thesis, we present novel algorithms to addressing various computational challenges in this field.

1.1 Genetics, DNA sequencing and Haplotyping

Genetics is the study of genes, genetic variation, and heredity in living organisms. Genetics controls what an organism looks like and how it functions. Specifically, there are two sides to the science of genetics. On the one hand, the availability of different types of molecular information, such as sequence information and gene expression levels, paired with gene editing techniques with which we can perturb the genome in a controlled fashion and observe its biological effects, provides powerful explanation tools to the functions of genes. On the other hand, genetics provides a fundamental understanding of how organisms, populations, and species evolve. In the last few years, one of the most exciting new developments is the way in which these two sides have begun to converge (Casillas and Barbadilla, 2017). This convergence is achieved through the development of technologies that provide datasets from the genomic level to epigenomic, transcriptomic or proteomics level.

The discovery of the double helix structure of deoxyribonucleic acid (DNA) by Watson and Crick in 1953 laid the main foundation for modern genetics. In most living organisms, genetic information is encoded in the form of DNA molecules. A DNA molecule is a chain in which many bases are ordered in a linear sequence, the bases — A, T, G, and C — are the letters of the genetic alphabet. The whole information within the DNA molecules of an organism is called its genome. The genome is divided into chromosomes. Genomes have single (haploid), double (diploid) or higher ploidy with more than two homologous chromosomes (polyploids). In this thesis, we focus on *diploid* organisms. For example, humans are diploids, consisting of two copies of each chromosome called homologous chromosomes or *haplotypes* — one inherited from the mother and the other from the father. There are differences between these two copies of each chromosome known as genetic variation.

In the early 2000s, a historic breakthrough happened with the sequencing of the human genome (Venter et al., 2001; Collins et al., 2003). Moreover, sequencing datasets of the human genome are made available as open-access to help in their interpretation and understanding (Church, 2005). Specifically, *sequencing* is an operation that determines the base sequence of a DNA molecule. For sequencing

genomes, there exist several kinds of sequencing technologies which share the following common properties. First, they yield genomes in fragments called "reads". Second, the position of reads along the genome sequence is unknown and also, mostly, the strand from which the read was sampled is unknown. Third, the reads contain errors.

Sequencing technologies differ in terms of error rates and lengths of the produced reads. We define the error rate of a read as the ratio of the number of incorrectly sequenced bases to the length of the read. Broadly, the sequencing technologies can be categorized into three classes:

- First generation sequencing: The first sequencing technologies were developed in 1977 by Sanger et al. (1977), who is awarded a Nobel Prize in chemistry, and Maxam and Gilbert (1977). Sanger sequencing produces reads in the order of 800-1000 base pairs with an error rate $\leq 1\%$.
- Second generation sequencing: It includes Illumina/Solexa sequencing (Bentley et al., 2008), which has been the most widespread technology. This technology produces short reads (hundreds of bases) with an error rate $\leq 1\%$.
- Third generation sequencing: It includes Single Molecule Real Time sequencing (PacBio) (Eid et al., 2009) and Oxford Nanopore sequencing (ONT) (Laszlo et al., 2014). These technologies produce very long sequences that are up to hundreds of kilo-bases in length. The downside is that PacBio and ONT exhibit very high error rates of up to 15% and 38% respectively.

Furthermore, there exist some sequencing protocols that provide long-range information. One of the sequencing protocols is the Strand specific protocol (Strand-Seq), which includes preparation of single-cell libraries. Also, each single strand of a DNA molecule is labeled regarding its 5'–3' orientation (Falconer et al., 2012). Illumina sequencing is performed from parental template strands in these single-cell libraries. The sequencing results in Illumina reads, along with information on the directionality of DNA. The other sequencing protocol is the 10x Genomics protocol (Eisenstein, 2015), which adds a unique barcode to every short read (produced from Illumina platform) generated from an individual molecule.

Sequencing data is routinely used to reconstruct the underlying haplotypes for diploid genomes. Reconstructing haplotypes is required in order to correctly understand allele-specific expression and compound heterozygosity. Compound heterozygosity is the phenomenon when the two homologous copies of a genomic region each contain unique sequence variants, but at different positions in that region. These variants are responsible for different functioning of the two homologous copies of a gene, resulting in different phenotypes. Thus, in settings in which compound heterozygosity play a role, the knowledge about the specific haplotype is essential. Haplotypes also help in investigating the genetic determinants of common diseases, and in performing population-genetic analyses of admixture, migration and selection (Tewhey et al., 2011; Glusman et al., 2014). Furthermore, haplotype sequences are used in relating genotypes to phenotypes, and for understanding how the arrangement of cis- and trans-acting variants across the two homologous copies of a genomic region affects phenotypic expression.

Technological progress in sequencing and computational approaches has enabled the reconstruction of underlying haplotypes for diploid genomes. However, there are intrinsic benefits and challenges when utilizing sequencing reads from different sequencing technologies. Specifically, upcoming PacBio and ONT sequencing deliver long reads, but have high sequencing error rates. In contrast, Illumina sequencing delivers reads with low error rates, but have short lengths. Currently, no sequencing technology delivers data that is complete and non-erroneous. Thus, the reconstruction of accurate and complete haplotypes (without gaps) from sequencing datasets remains a challenging problem.

From available sequencing datasets, the first challenge for reconstructing haplotypes arises from a lack of information of the genomic location of reads. The next challenge is the lack of the *haplotypic identity* of reads, i.e. the haplotype that the read comes from. Knowing the haplotype of reads is essential for reconstructing both copies of each chromosome, which in turn helps better understand the true biological characteristics of diploid organisms.

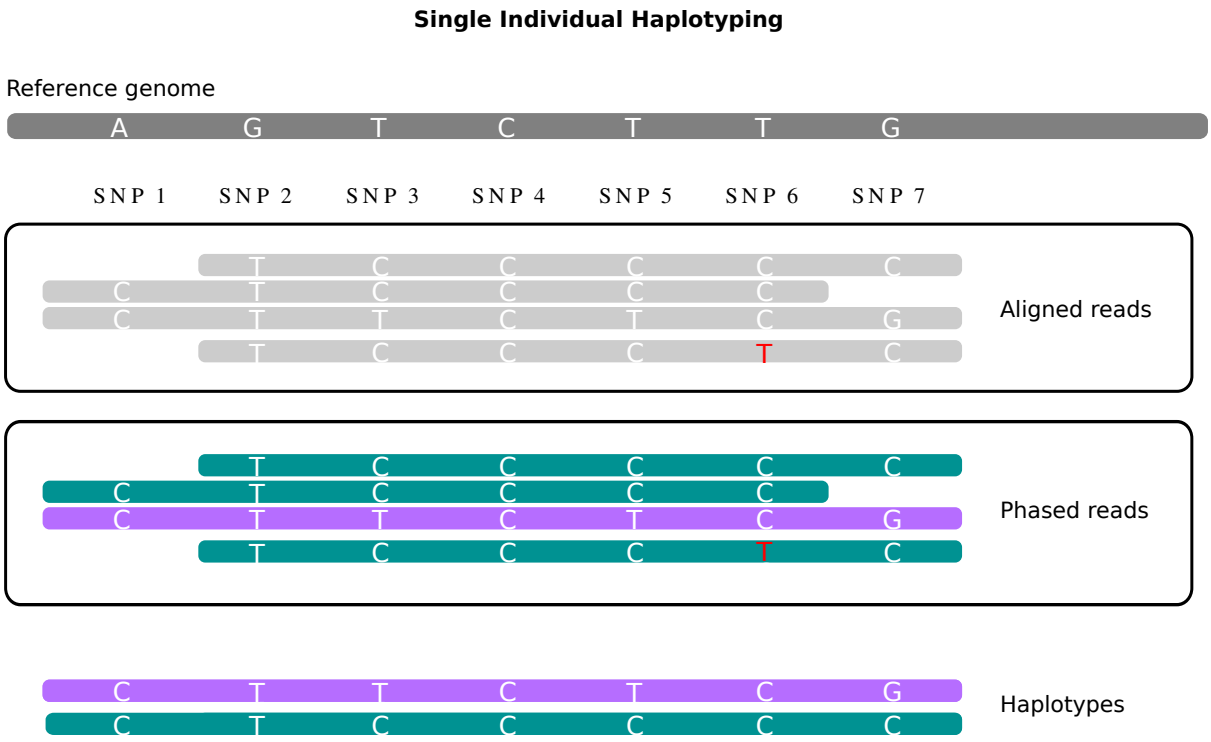

Figure 1.1: Seven variants covered by reads (horizontal bars) in a single individual. The alleles that a read supports are printed in white. The middle panel shows the phased reads in colors and haplotypes at the bottom over the seven variants.

The clever approaches are developed to solve these challenges. Broadly, the approaches to obtain haplotypes are classified into two categories: *reference-based* haplotyping and *haplotype-aware de novo* assembly.

1.2 Reference-based Haplotyping

Reference-based phasing methods are applied when a reference genome is available for the species of the target genome. We expect the target genome to be very close to the reference, more specifically, given the reads of the target genome, we expect the reference to be from the same species as the reads.

Reference genomes provide reliable information on the genomic location of reads. Over the years, a lot of effort has been devoted to generating good quality reference genomes (Consortium et al., 2010, 2005). Reference genomes provide the organization of the genome, including the relative position of genes or chromosomes structure (Consortium et al., 2004). Reference genomes have been used by biologists for other tasks, for instance, finding the functions to annotate the genome (Harrow et al., 2012).

Standard pipelines for reference-based phasing consist of the following steps: First, the reads are aligned to the reference genome. Second, the variants are detected using various variant calling algorithms, to finding the differences of target genome to the reference. Third, the detected variants are phased based on how aligned reads connect the alleles over them, to generate two haplotypes.

Please note that the different terminologies *reference-based* haplotyping (phasing), single individual haplotyping and read-based phasing are used interchangeably in this thesis.

To illustrate the haplotyping problem for a single genome, we consider a small example in Figure 1.1. The example shows seven variants. Also shown are the sequencing reads aligned to the reference genome. The alleles that the reads support are shown in white. Erroneous alleles in the reads are shown in red. In practice, we do not know the alleles that contain these sequencing errors. The goal

of the *reference-based* haplotyping problem is the re-assignment of phases to the reads, i.e. assigning haplotype-specific colors (green or purple) to each read. In the middle panel, the colored bars represent the assignment of each read to either the green or the purple haplotype. Finally, the reads from each haplotype are separately assembled together in order to output two haplotypes that are shown at the bottom in purple and green.

1.2.1 Haplotyping as a Combinatorial Optimization problem

We will see how we can formulate the haplotyping problem as combinatorial optimization problem, that is, given an object o, find a solution such that an optimization criterion f is either minimized or maximized.

For the haplotyping problem, which consists of determining the *haplotypic identity* of each read, we consider the reads that are aligned to reference genome. A read aligner maps the reads to the reference genome, ideally to positions with a high similarity score for the read. The number of read alignments that cover a position is known as the coverage for that position. Furthermore, we have structural nucleotide variants (SNVs) detected using different variant calling algorithms. In the case of bi-allelic variants, that is, those variants for which two different alleles are known, three genotypes are possible. The reference allele is typically denoted as 0 and the alternative allele as 1. Using this notation, the two chromosomal copies either both carry the reference allele (genotype 0/0), or alternative allele (genotype 1/1) or one of them contains the reference while the other one carries the alternative allele (genotype 0/1). If both chromosomal copies carry the same allele (i.e. genotype 0/0 or 1/1), the genotype is called homozygous, while genotype 0/1 is referred to as heterozygous. s Given the variants and the alignments, the goal here is to phase the variants and generate the haplotypes.

Mathematically, the aligned reads over the variants are encoded in the form of a SNP matrix. The SNP matrix for the example given in Figure 1.1 is illustrated in Figure 1.2. The SNP matrix $\mathcal{F}$ is an element of $\{0, 1, -\}^{R\times M}$, where R is the number of reads and M is the number of variants along a chromosome. Each matrix entry $\mathcal{F}(j, k)$ is 0 (indicating that the read matches the reference allele) or 1, indicating that the read matches the alternative allele if the read covers that position and "–" otherwise. Note that the "–" character can also be used to encode the unsequenced "internal segment" of a paired-end read. The goal of the haplotype assembly problem is to generate two haplotypes $h^0, h^1 \in \{0, 1\}^M$.

The presence of sequencing and mapping errors makes the haplotype assembly problem a challenging task. Computationally, this problem has been generally modeled as an optimization problem to correct the sequencing and mapping errors. In literature, different combinatorial formulations of the problem have been proposed (Lippert et al., 2002). Among them, Minimum Error Correction (MEC) (Lippert et al., 2002) has been proven particularly successful in the reconstruction of accurate haplotypes for diploid species (Martin et al., 2016; He et al., 2010; Chen et al., 2013; Glusman et al., 2014; Rhee et al., 2016). MEC aims to correct the input data with the minimum number of corrections to the SNP values, such that the resulting reads can be partitioned into two sets, with each set identifying a haplotype. To mathematically formulate the minimum number of corrections (MEC) as an optimization problem, we require a few definitions.

The quality of a solution relies on the measure $d(r_1, r_2)$ based on the Hamming distance between any two rows $r_1, r_2 \in \{0, 1, -\}^M$ in $\mathcal{F}$.

Definition 1.1 (Distance). Formally, the distance between rows r_1 and r_2 is given by

$$d(r_1, r_2) := \left|\left\{k \,\middle|\, r_1(k) \neq - \ \wedge \ r_2(k) \neq - \ \wedge \ r_1(k) \neq r_2(k)\right\}\right|.$$

Definition 1.2 (Feasibility). A SNP matrix $\mathcal{F} \in \{0, 1, -\}^{R\times M}$ is called *feasible* if there exists a bi-partition of rows (i. e., reads) into two sets such that all pairwise distances of two rows within the same set are zero.

Feasibility of a matrix $\mathcal{F}$ is equivalent to the existence of two haplotypes $h^0, h^1 \in \{0, 1\}^M$ such that every read r in the matrix has a distance of zero to h^0 or to h^1 (or both). The MEC problem can now

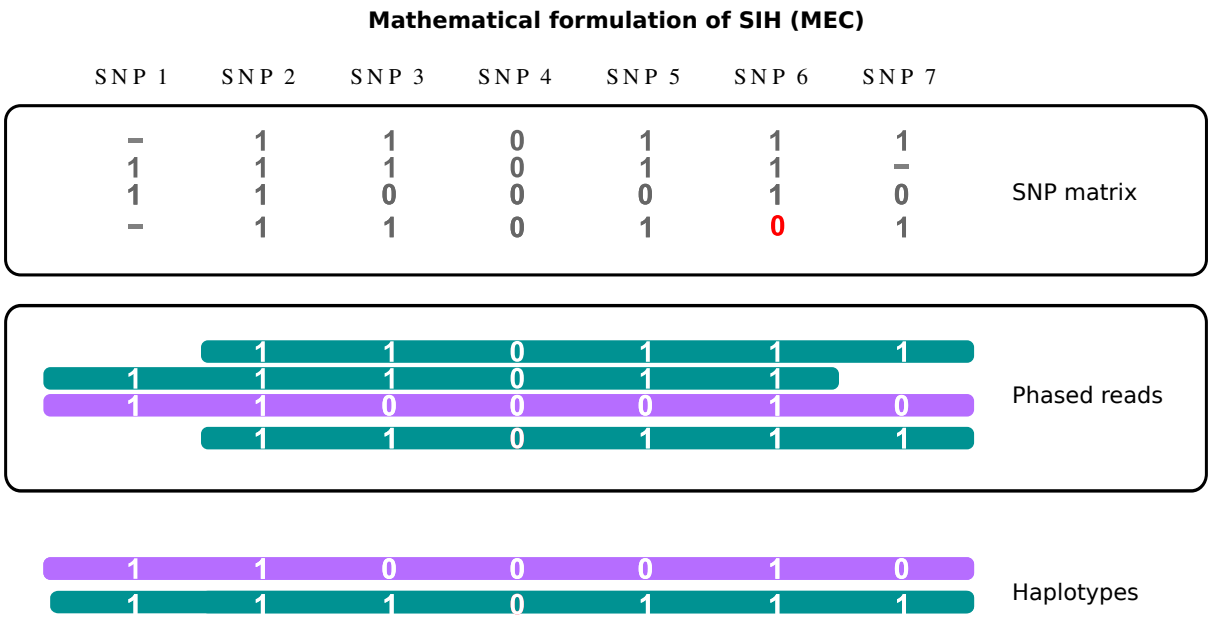

Figure 1.2: Example shows the SNP matrix for the example shown in Fig. 1.1. Seven variants covered by reads (horizontal bars) in a single individual. The allele in read is encoded as 1 if it matches the allele in the reference position at that position and 0 otherwise. The middle panel shows the phased reads in colors and haplotypes at the bottom over these seven variants.

simply be stated in terms of flipping bits in $\mathcal{F}$, where entries that are 0 or 1 can be flipped and "–" entries remain unchanged.

Problem 1.1 (MEC). Given a matrix $\mathcal{F} \in \{0, 1, -\}^{R \times M}$, flip a minimum number of entries in $\mathcal{F}$ in order to obtain a feasible matrix.

Definition 1.3 (MEC cost). The MEC cost for a solution $h^0, h^1 \in \{0, 1\}^M$ is given by:

$$\text{cost}_{\mathcal{F}}(h^0, h^1) := \sum_{i=1}^{n} \min\{\text{dist}(r_i, h^0), \text{dist}(r_i, h^1)\}$$

.

where $r_i \in \{0, 1, -\}^M$ is the i-th row of a SNP matrix $\mathcal{F}$.

The MEC problem is NP-hard (Cilibrasi et al., 2007), but more detailed analyses require a more fine grained distinction between different instances types, which are described as follows.

- MEC: Instances in which entries in each of the n rows of $\mathcal{F}$ are from $\{0, 1, -\}$. There is no restriction on the placement of entries 0, 1 and −. These instances are generated from Illumina, 10x Genomics and Strand-Seq sequencing technologies.
- Gapless-MEC: A MEC instance is called *gapless* if the entries in each of the n rows of $\mathcal{F}$ follows a regular expression of the type $-^*\{0, 1\}^*-^*$. These instances are generated from PacBio, ONT and single-end Illumina like technologies.
- Binary-MEC: Instances in which entries in each of the n rows of $\mathcal{F}$ are from $\{0, 1\}$ with no gaps.

Figure 1.3 shows an example of a mathematical representation of reads from different sequencing technologies, that cover seven variants. The top panel shows a general Binary-MEC instance consisting of binary values with no gaps, the middle panel shows a Gapless-MEC instance with binary values in between and gaps at its two ends and, the bottom is a MEC instance which consists of binary values and gaps with no restriction on placement of gaps.

Additionally, we consider a weighted version of the MEC (wMEC), in which a cost is associated to every matrix entry. This is useful in practice since each nucleotide in a sequencing read usually comes with a "phred-scaled" base quality Q that corresponds to an estimated probability of $10^{-Q/10}$ that

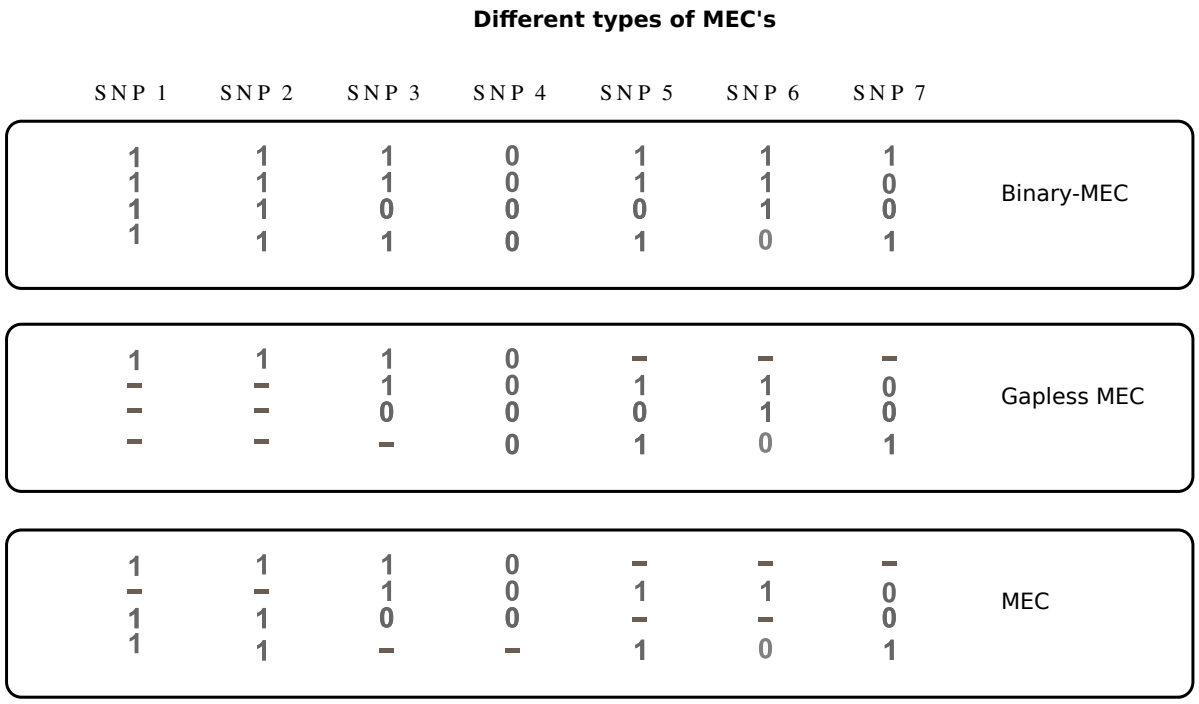

Figure 1.3: Seven variants covered by reads (horizontal bars) in a single individual are represented as MEC instances. At the top is a general MEC instance with arbitrary gaps, the middle is a Gapless-MEC instance with gaps only at its two ends and the bottom is a Binary-MEC instance which consists of only binary values.

this base has been wrongly sequenced. These phred scores can hence serve as costs of flipping a letter, allowing less confident base calls to be corrected at a lower cost compared to high confidence ones.

Problem 1.2 (wMEC). Given a matrix $\mathcal{F} \in \{0, 1, -\}^{R \times M}$ and a weight matrix $\mathcal{W} \in \mathbb{N}^{R \times M}$, flip entries in $\mathcal{F}$ to obtain a feasible matrix, while minimizing the sum of incurred costs, where flipping entry $\mathcal{F}(j, k)$ incurs a cost of $\mathcal{W}(j, k)$.

Beyond the MEC formulation and its variants, other objective functions as surveyed by Rhee et al. (2016) have been proposed.

The other objective functions to solving haplotype assembly problem are as follows:

- Minimum fragment removal (MFR) and its weighted version (WMFR): These objective functions derive the haplotype assembly by removing rows of the matrix $\mathcal{F}$.
- Minimum SNP removal (MSR) and its weighted version (WMSR): These objective functions derive the haplotype assembly by removing columns of the matrix $\mathcal{F}$.
- Minimum fragment cut (MFC): This function involves partitioning of the rows of $\mathcal{F}$ into two segments representing haplotypes.
- Other objective functions such as Graph and Satisfiability (SAT) formulations.

The focus of the thesis is on the Minimum Error Correction (MEC) formulation for solving the haplotype assembly problem.

1.2.2 Related work

Next, we survey existing algorithmic approaches, mainly focused on the Minimum Error Correction formulation, to solving the haplotype assembly problem, both in theory and practice.

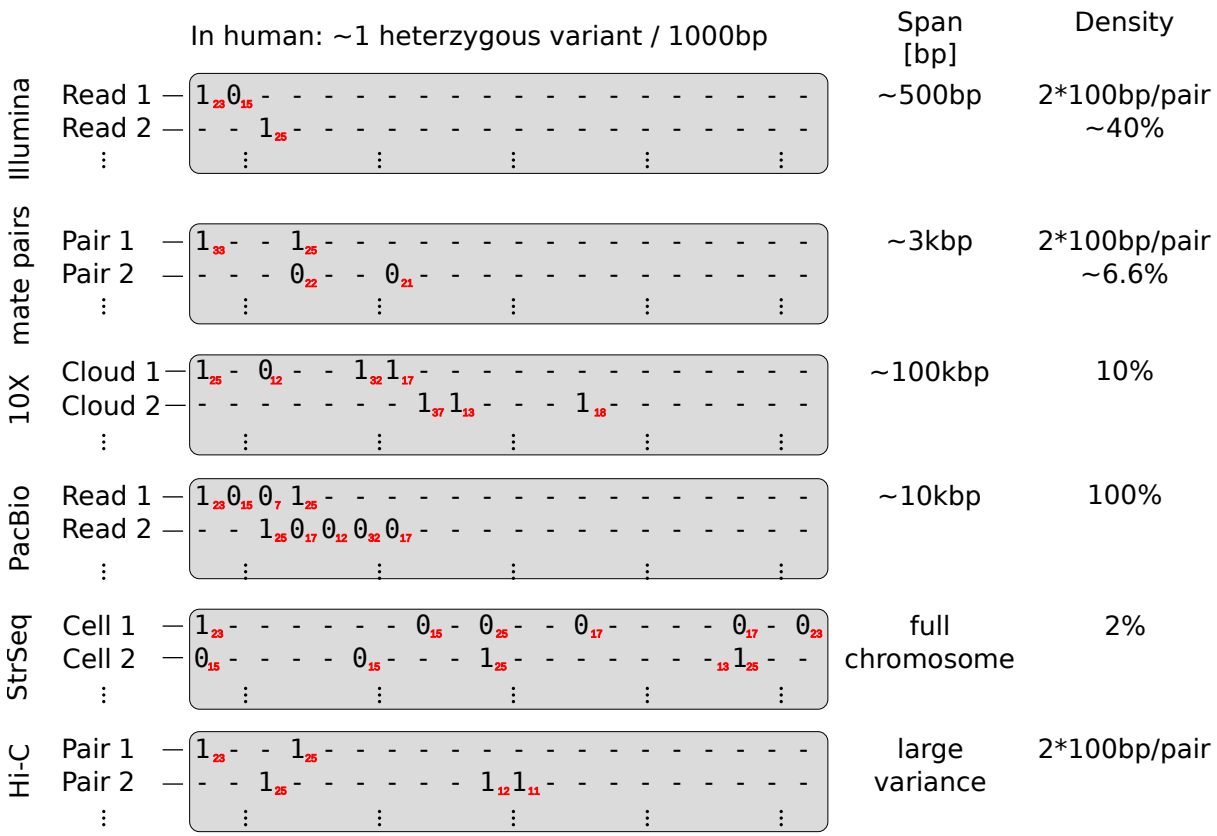

Figure 1.4: Variants covered by reads in a single individual are represented as MEC instances from different sequencing technologies. The weights are shown in red. Figure from a paper by Klau and Marschall (2017).

1.2.2.1 Theoretical approaches

In this section, we discuss theoretical approaches for the MEC and its versions for solving the haplotyping problem. For the simplest version, Binary-MEC, finding its optimal solution is equivalent to solving the hypercube 2-segmentation problem (H2S). The H2S problem was introduced by Kleinberg et al. (1998, 2004) and is known to be NP-hard (Feige, 2014; Kleinberg et al., 2004). The optimization version of Binary-MEC is closely related to H2S. In the former we minimize the number of mismatches instead of maximizing the number of matches. In particular, the NP-hardness of H2S directly implies the NP-hardness of Binary-MEC, Gapless-MEC, and MEC. Gapless-MEC and MEC were shown to be NP-hard by Cilibrasi et al. (2007).[1]

With respect to the approximation status, Ostrovsky and Rabani (2002) obtained a polynomial time approximation scheme (PTAS) (see Definition 2.5) for Binary-MEC based on random embeddings. Building on the work of Li et al. (2002), Jiao et al. (2004) presented a deterministic PTAS for Binary-MEC. For Gapless-MEC, a logarithmic factor approximation is the best known approximation algorithm so far (Bonizzoni et al., 2016). Furthermore, the MEC problem in its general form is known to be APX-hard (Cilibrasi et al., 2007). Therefore, there is a gap on the approximation status of Gapless-MEC: it is unknown whether it is as easy as Binary-MEC or as hard as MEC.

Open Problem 1.1. The approximation status of Gapless-MEC is an open problem; Gapless-MEC instances are important because they are often produced by single-ended PacBio or ONT technologies. Deriving polynomial time approximation algorithms (PTAS) for Gapless-MEC instances provides evidence that Gapless-MEC can be solved in polynomial time.

[1] Their result predates the hardness result of Feige (2014) for H2S. The proof of the claimed NP-hardness of H2S by Kleinberg et al. (1998) was never published.

1.2.2.2 Approaches in practice

Here, we discuss exact as well as heuristic approaches, which are often used in practice for solving the haplotype assembly problem in a time efficient manner.

Exact approaches. The exact approaches, which solve the problem optimally, include integer linear programming (Fouilhoux and Mahjoub, 2012; Chen et al., 2013), and fixed-parameter tractable (FPT) algorithms (He et al., 2010; Patterson et al., 2015; Pirola et al., 2015).

An *Integer Linear Program (ILP)* consists of two parts, constraints or conditions and an objective function. Additionally, the objective function and the constraints are linear. An ILP in standard form can be expressed as

$$\begin{aligned} \text{maximize} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} + \mathbf{s} = \mathbf{b}, \\ & \mathbf{s} \geq \mathbf{0}, \\ \text{and} \quad & \mathbf{x} \in \mathbb{Z}^n, \end{aligned}$$

where $\mathbf{c}$, $\mathbf{b}$, $\mathbf{x}$ and $\mathbf{s}$ are vectors, A is a matrix and entries in $\mathbf{x}$ are integers.

Chen et al. (2013) proposed an ILP-based approach to solving the MEC problem. They consider binary variables for each row and column and their corresponding values are supposed to be dependent on haplotypes.

Branch-and-Bound algorithm. Branch-and-bound algorithms enumerate the candidate solutions by using a rooted tree. The algorithm explores the branches of this tree, which represent subsets of the solution set. Before enumerating the candidate solutions of a branch, the branch is compared to an upper and lower estimated bounds on the optimal solution, and is ignored if it cannot produce a better solution than the best one found so far by the algorithm. Wang et al. (2005) applied a branch-and-bound algorithm for solving the MEC to finding haplotypes. This approach solves the problem in an exact way, but does not scale well for large datasets.

Parameterized algorithms. Parameterized algorithms choose a fixed parameter and solve a problem in time exponential only in the size of this fixed parameter, but polynomial in the input size. Such an algorithm is called a fixed-parameter tractable (FPT) algorithm, because problem instances can be solved efficiently for small values of the fixed parameter.

In NGS data analysis, there are several parameters, such as read length, coverage and number of sequencing errors, that can help in solving genomics problems efficiently. Choosing a parameter, that is small enough to work in practice, is an art. In the works by He et al. (2010); Patterson et al. (2015); Pirola et al. (2015); Martin et al. (2016); Klau and Marschall (2017), different parameters are proposed to solve the MEC problem with FPT approaches. These approaches are applied on each NGS dataset separately, without considering these datasets in combination.

As described in Section 1.1, the NGS datasets have their inherent challenges and benefits and, no technology independently enables producing complete haplotypes. For instance, long-read technologies have high sequencing error rates and short-read technologies produce short reads, which are not sufficient to producing the whole genome. Furthermore, the Strand-Seq technology produce long-range information, but provide sparse information. These limitations from different technologies necessitates to combine the advantages of different technologies in a joint optimization framework, in order to facilitate producing complete haplotypes. In the recent work by Edge et al. (2017), they assemble haplotypes by combining multiple sequencing technologies such as SMRT (PacBio) sequencing, linked-read sequencing and proximity litigation sequencing. However, an efficient algorithm to solve haplotyping, by combining Strand-Seq sequencing datasets with other sequencing technologies in an integrative framework, is unknown. Therefore developing a parameterized algorithm for this integrative framework and deciding parameters that work well in practice is very important.

The corresponding MEC instances for a single genome from different technologies such as Illumina, PacBio and Strand-Seq technologies are shown in Figure 1.4. Shown is the toy example of MEC instances for a human genome that consists of heterozygous variants, one in every 1000 bp. Over these variants,

we observe that the matrix from Illumina data contains more gaps compared to binary values in every row because of the short read nature. The matrix from long-read technologies such as PacBio and ONT contain more binary values in every row. The matrix from Strand-Seq data is very sparse with more gaps, and has arbitrary positioning of gaps and binary values. The reads from any technology independently do not fill all the columns in the matrix with binary values and therefore, cannot generate end-to-end haplotypes. The joint matrix from different combinations of technologies contains binary values in all the columns and thus can produce end-to-end haplotypes.

Open Problem 1.2. Finding a parameterized algorithm for a version of MEC that uses multiple sequencing technologies in an integrative framework, is an open problem.

Heuristic approaches. A heuristic algorithm is one that is designed to solve a problem in an efficient fashion in terms of speed and memory requirements, at the cost of sacrificing optimality. Heuristic algorithms are most often employed when sub-optimal solutions are sufficient and exact solutions are computationally expensive.

Many different heuristic approaches have been proposed for haplotyping. HASH (haplotype assembly for single human) uses a Markov chain Monte Carlo (MCMC) algorithm and a graph partitioning approach to assemble haplotypes, given a list of heterozygous variants and a set of shotgun sequence reads mapped to a reference genome assembly (Bansal et al., 2008). In the work by Wang et al. (2007), a clustering algorithm is deployed to split the rows of $\mathcal{F}$ in two sets based on MEC formulation. HapCut (Bansal and Bafna, 2008) utilizes the overlapping structure of the fragment matrix and max-cut computations to find the minimum error correction (MEC) solution for haplotype assembly. Duitama et al. (2010) follows a heuristic approach for max-cut to find haplotypes efficiently. MixSIH (Matsumoto and Kiryu, 2013) utilizes a probabilistic mixture model to solve haplotyping. H-BOP (Xie et al., 2012) follows a heuristic algorithm for optimizing a combination of the MEC and Maximum Fragments Cut models. ProbHap (Kuleshov, 2014) employs a similar approach to Patterson et al. (2015), but ProbHap uses the Viterbi algorithm to solve the maximum likelihood function specified by a probabilistic graphical model.

1.2.3 Pedigree of genomes

Another approach to haplotyping takes into account the sequencing datasets from multiple members of a families. Specifically, such an approach takes advantage of both sequencing data of each individual and of the principles of the Mendelian segregation. These are highly informative for identifying haplotypes that are identical-by-descent (IBD) between individuals within a pedigree. At the simplest level of a family trio (both parents and one child), simple rules indicate which alleles in the child were inherited from each parent, thus largely separating the two haplotypes in the child. This process of separating the two haplotypes based on genotype data alone is called genetic haplotyping. Nevertheless, genetic haplotyping cannot phase positions in which all family members are heterozygous. In such cases using sequencing dataset can complement the genetic haplotyping.

The Haploscribe method (Roach et al., 2011) phased whole-genome data based on genetic haplotyping. Haploscribe followed a parsimony approach to generate meiosis-indicator (inheritance state) vectors and obtained haplotypes by modeling the haplotyping problem using a hidden Markov model (HMM). Abecasis et al. (2002) is based on genetic information that formed clusters of tightly-linked sites within linkage disequilibrium. Williams et al. (2010) followed a dynamic programming approach to compute the maximum likelihood haplotypes from genotype data of pedigrees.

All these previous approaches lack the joint usage of both information sources based on IBD and sequencing datasets of individuals in the pedigree. Finding an efficient method that uses both sequencing data and genetic inheritance principles in an integrative fashion for performing phasing is important for generating complete haplotypes.

Open Problem 1.3. Combining information pertaining to genetic inheritance, and sequencing reads into one framework is an open problem. Furthermore, it is not clear whether it is possible to design an efficient algorithm that works well in practice is an important question.

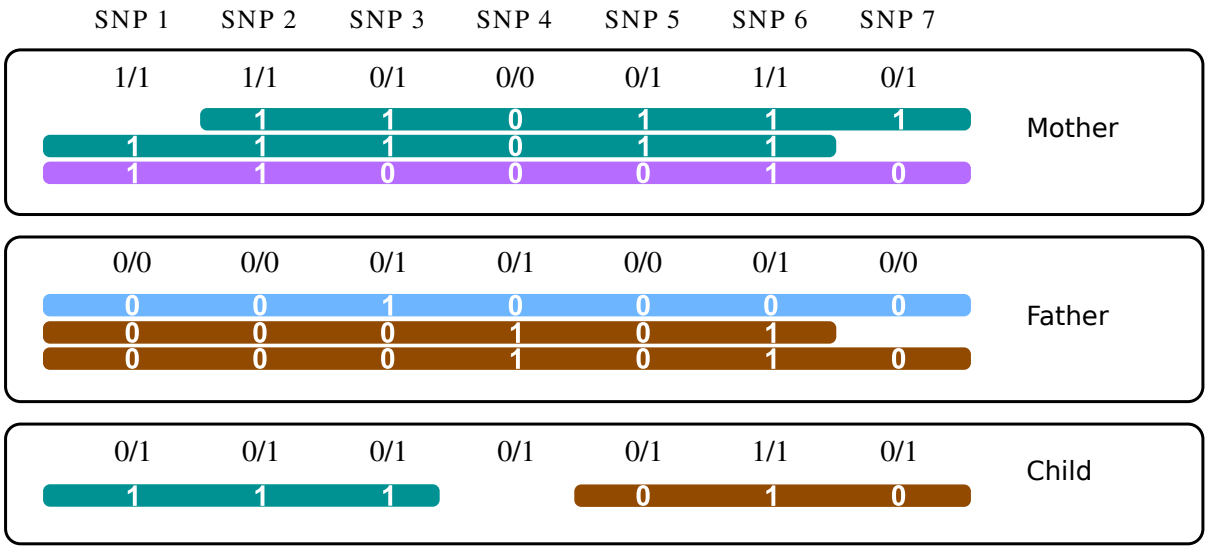

Figure 1.5: Seven SNP loci covered by reads (horizontal bars) in three individuals. Unphased genotypes are indicated by labels 0/0, 0/1 and 1/1. The alleles that a read supports are printed in white.

To illustrate the motivation to combine genetic and read-based haplotyping, the corresponding MEC instance for a single genome is shown in Figure 1.5. There are seven SNP positions covered by reads in three related individuals. This illustrates how the ideas of genetic and read-based haplotyping complement each other. All genotypes at SNP 3 are heterozygous. Thus, its phasing cannot be inferred by genetic phasing, that is, using only the given genotypes and not the reads. SNP 4, in contrast, is not covered by any read in the child. When only using reads in the child (corresponding to single-individual read-based phasing), no inference can be made about the phase of SNP 4 and neither about the phase between SNP 3 and SNP 5. The phases of all SNPs for the child can be inferred based on the observation that all seven child genotypes are compatible with the combination of brown and green haplotypes from the parents. This example demonstrates that using pedigree information, genotypes and sequencing reads jointly is very powerful for establishing phase information.

1.2.4 Statistical phasing

Another approach to haplotyping includes inferring haplotypes from genotype information of large cohorts based on the idea that common ancestry gives rise to shared haplotype tracts, as reviewed by Browning and Browning (2011); Loh et al. (2016b,a). This approach is known as *statistical* or *population-based phasing*. Popular tools based on statistical phasing approach are Beagle (Browning and Browning, 2007), ShapeIt (Delaneau et al., 2013a,b) and Eagle (Loh et al., 2016b,a).

This approach can be applied to unrelated individuals and only requires genotype data, which can be measured at low cost. While very powerful for common variants, this technique is less accurate for phasing rare variants and cannot be applied at all to private or *de novo* variants.

In the above section, we have presented an overview of computational approaches for performing reference-based haplotyping by using the reference genome as a backbone. Relying on a reference genome may hinder the correct analyses in some cases. First, read mapping step in this method has a reference bias because the reference genome does not capture the genomic diversity of a population. The reads that are unique to the target genome, are not aligned or wrongly aligned to the reference genome. The usage of reference genomes for read alignment hence generate a bias. Second, we make the prior hypothesis that the target genome is close to the reference, which may not always be true in reality. Third, the method is not self-sufficient since a prior reference needs to be constructed. For these reasons, we additionally consider *haplotype-aware de novo* (without reference) or *diploid* assembly.

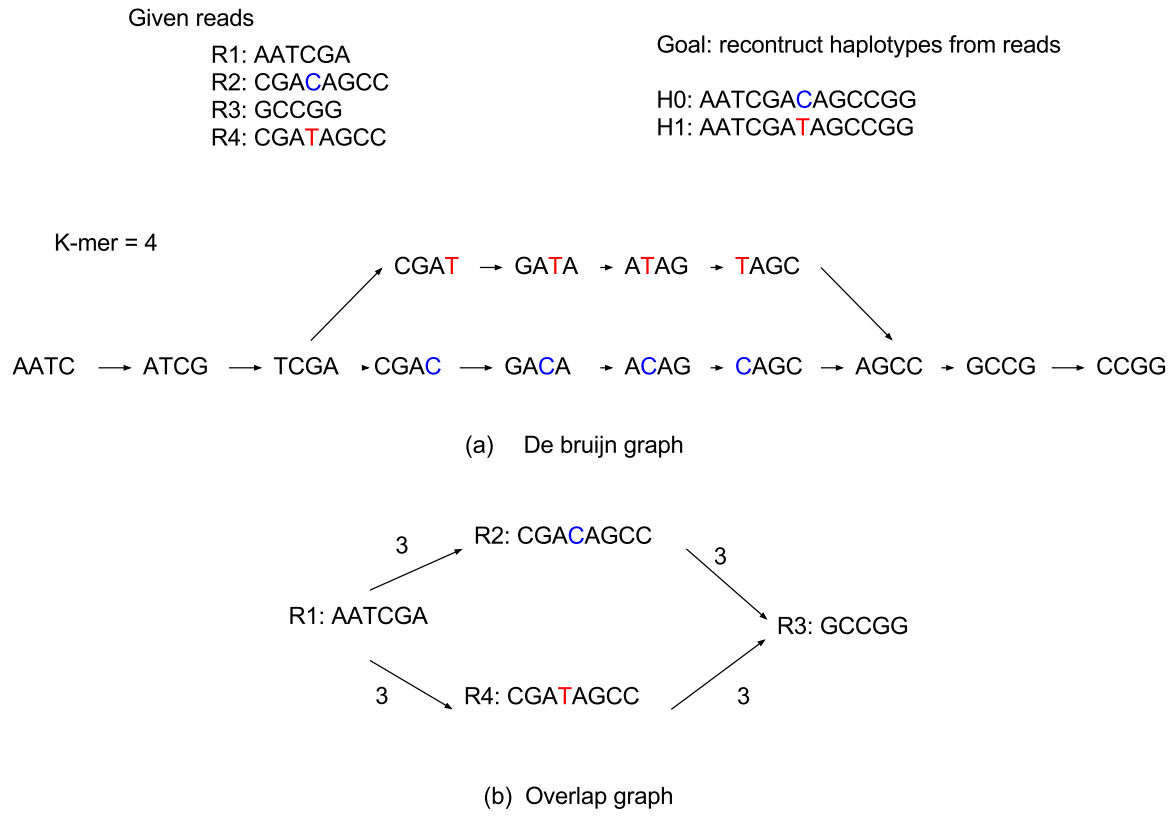

Figure 1.6: Figure shows the reads and reconstructed haplotypes using two graph approaches: (a) de Bruijn graph and (b) overlap graph.

1.3 Diploid assembly

In diploid assembly, the reads from a diploid genome are directly used to assemble the haplotype sequences. The diploid assembly process involves partitioning the input set of reads into two disjoint sets, and then gluing together the reads from each set in proper order to produce diploid assemblies. Therefore, diploid assembly process aims to infer the *ordering* and *haplotypic* identity of input sequencing reads. The diploid assembly process is challenging due to short read lengths, incomplete data, sequencing errors, and repetitive regions on the genome.

Below, we discuss how to formulate the diploid assembly problem of finding the ordering and haplotypic identity of reads for producing diploid assemblies, while avoiding misassemblies in complex repetitive regions.

1.3.1 Diploid assembly as a graph problem

The diploid assembly from sequencing reads is modeled as the graph problem. The assembly graph restores reliable information about the *ordering* of reads. Assembly graphs can be categorized into two families: overlap graphs and de Bruijn graphs.

Overlap graphs. Given a set of reads, the overlap graph consists of nodes that represent reads, and edges that represent the overlap between read sequences. The weight on the edges represents the maximal overlap length between two sequences. As illustrated in Figure 1.6b, given four reads, the goal is to reconstruct the haplotypes sequences H0 and H1. In the shown overlap graph, there are four nodes

R1, R2, R3 and R4 for these reads and there are edges between these nodes based on the overlap, for example, there is an edge between R1 and R2 with a weight of 3.

Overlap graphs can be simplified to string graphs by the transitive reduction of edges (Myers, 2005). Also, contained edges (Myers, 2005) are removed, which occur when one read is a substring of other reads.

Most of the assemblers generate only one sequence (consensus sequence) and the algorithm to construct this sequence (instead of two) can be outlined as follows.

- Overlap: calculate pairwise overlaps between reads
- Layout: compute a parsimonious solution (as a generalized Hamiltonian path visiting each node at least once while minimizing the total string length)
- Consensus: merge reads, using redundancy to correct sequencing errors

The first OLC assembler was developed by Celera (Myers et al., 2000), which was designed to handle sequencing data from Sanger technology. Celera employs a BLAST-like approach for performing all-vs-all read alignment. Celera then compacts the overlaps with no ambiguity and uses some heuristics on the tangled regions that arise due to repeats. The final sequences are generated by forming a consensus between reads to sequencing errors. Complex repetitive regions are hard to resolve, resulting in fragmented assemblies. We call these fragmented sequences "contigs" for contiguous consensus sequences. Furthermore, a series of contigs are connected using long-range read information such as Hi-C and optimal mapping information, to generate long assemblies called "scaffolds".

This paradigm was used with long Sanger sequences and for relatively small genomes. Currently, the OLC-based algorithms are also used with PacBio datasets (Grohme et al., 2018).

De Bruijn graphs. The de Bruijn graph is a directed graph representing overlaps between sequences of symbols, named after Nicolass Govert de Bruijn (Todd, 1933). In this type of assembly graph, each read is broken into a sequence of overlapping k-mers, where a k-mer is a substring of length k. The distinct k-mers are added as vertices to the graph, and k-mers with $k-1$ overlap are connected by an edge. In Figure 1.6a, the reads are divided into words of fixed length k, where $k = 4$. Here, each node in the graph is a word and the connections between the nodes are based on the overlap between nodes.

The first application of the de Bruijn graph in genome assembly was introduced in the EULER assembler (Pevzner et al., 2001). The assembly problem can then be formulated as finding a walk through the graph that visits each edge in the graph once — an Eulerian path problem. Due to the repeats, it is difficult to find Euler paths. In most instances, the assembler attempts to construct contigs consisting of the unambiguous, unbranching regions of the graph.

Medvedev et al. (2007) extended the original directed de Bruijn graph model of Pevzner et al. (2001) to a bi-directed de Bruijn graph (BDDG) model, which is more efficient in handling DNA's double-strand structure. A bi-directed graph is a generalized directed graph in the sense that two end-points of an edge are given independent orientations (or directions) at each end. Thus, from sequencing reads $R = \{R_1, R_2, ...R_m\}$, we can generate a bi-directed de Bruijn graph by defining the vertices as k-mers obtained from the reads and then getting all possible edges among the vertices.

The advantage of BDDG over directed de Bruijn graph is simplification. Both strands of a k-mer are represented by one node in BDDG compared to two separated nodes in directed graph. For example, if forward strand is represent by node n, then the reverse strand is represented by n' within the same node n. This saves memory and storage but also simplifies the graph so that graph operations are more efficient.

De Bruijn graph and overlap graph. The de Bruijn graph theoretically achieves the same tasks that the overlap graph does, but in an efficient manner (Li et al., 2012). The de Bruijn graph became widely for assembling the short reads. The OLC approach did not scale well on the high number of sequences generated by NGS. The use of the de Bruijn graph is prevalent for short read assembly because these approaches employ efficient techniques to handle redundancy in data. Indeed a k-mer present multiple times in the sequencing dataset appears only once in the graph. This makes the de

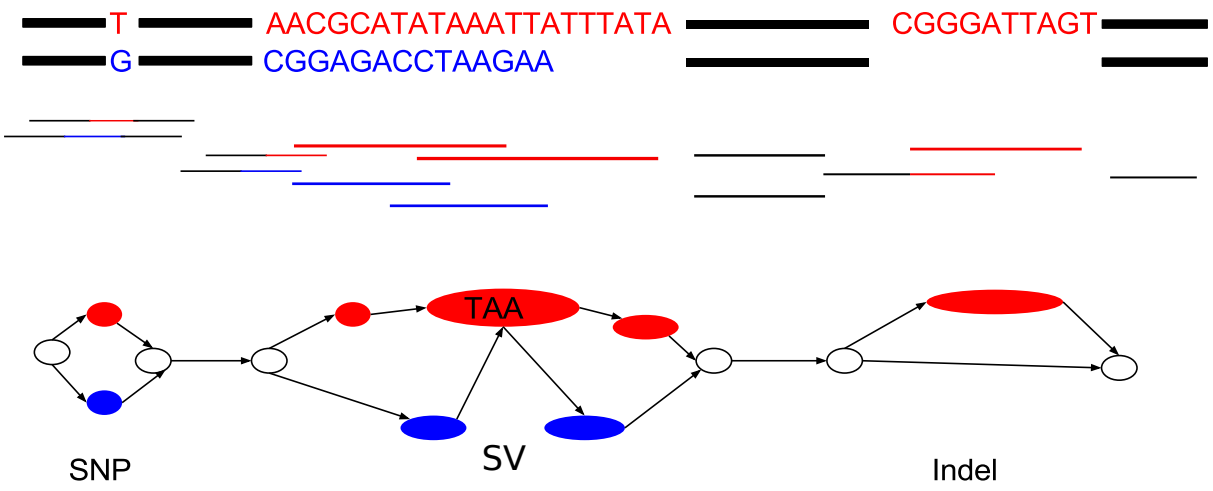

Figure 1.7: Given the input reads (middle) from the two sequences (top), we show a corresponding assembly graph at the bottom. The bubbles in the sequence graph (bottom) show three different heterozygous variations; the first one is an SNV, the second one is an SV, and the third one is an indel.

Bruijn graph structure not very sensitive to high coverage, unlike OLC. The de Bruijn graph was first proposed as an alternative structure (Pevzner et al., 2001) because it was less sensitive to repeats.

We now discuss the special structures in assembly graphs, that can occur due to repeats and heterozygous regions.

Graph structures. We discuss mainly two types of structures (bubbles and repeats) that occur in the assembly graphs for diploid genomes.

Bubbles. Bubbles are defined as a set of disjoint paths bounded by a fixed start and an end node and all paths through the bubble flow from start to end. No other vertex in the graph other than a start node forms a pair with an end node. Bubbles in the graph represent heterozygosity or sequencing errors for the diploid organism. Bubbles can contain simple SNVs with only one bp difference, or even large complex structural variations in the order of kilo-bases or more. Figure 1.7 illustrates how bubbles in an assembly graph can contain both small structural variants and large structural variants.

Repeats. Repeats in a genome causes branches or cycles in the assembly graphs and, therefore, make a graph more complex and break the properties of the linear reference. Specifically, it becomes difficult to find the positioning or linear ordering of nodes in the graph. The repeats in the graph are illustrated in Figure 1.8. In this example, the repeat R causes cycles and branches in the graph. Assemblers generally handle these repeats by making a greedy guess as to which branch to follow. Incorrect guesses create false joins (chimeric contigs) and erroneous copy numbers. If the assembler is more conservative, it will break the assembly at these branch points, leading to an accurate but fragmented assembly with fairly small contigs. The ability to resolve repeats depends on the reads length. If there is a read that is long enough to span the repeat region, then the repeat is resolvable. Therefore, upcoming long-read sequencing technologies produce reads that span these repeats helps in obtaining maximally repeat-resolved diploid assemblies.

1.3.2 Related work on diploid assembly

Over the last decade, the development of various NGS technologies has impacted the assembly problem. In theory, the problem of *de novo assembly*—computing the consensus of two or more sequences—is

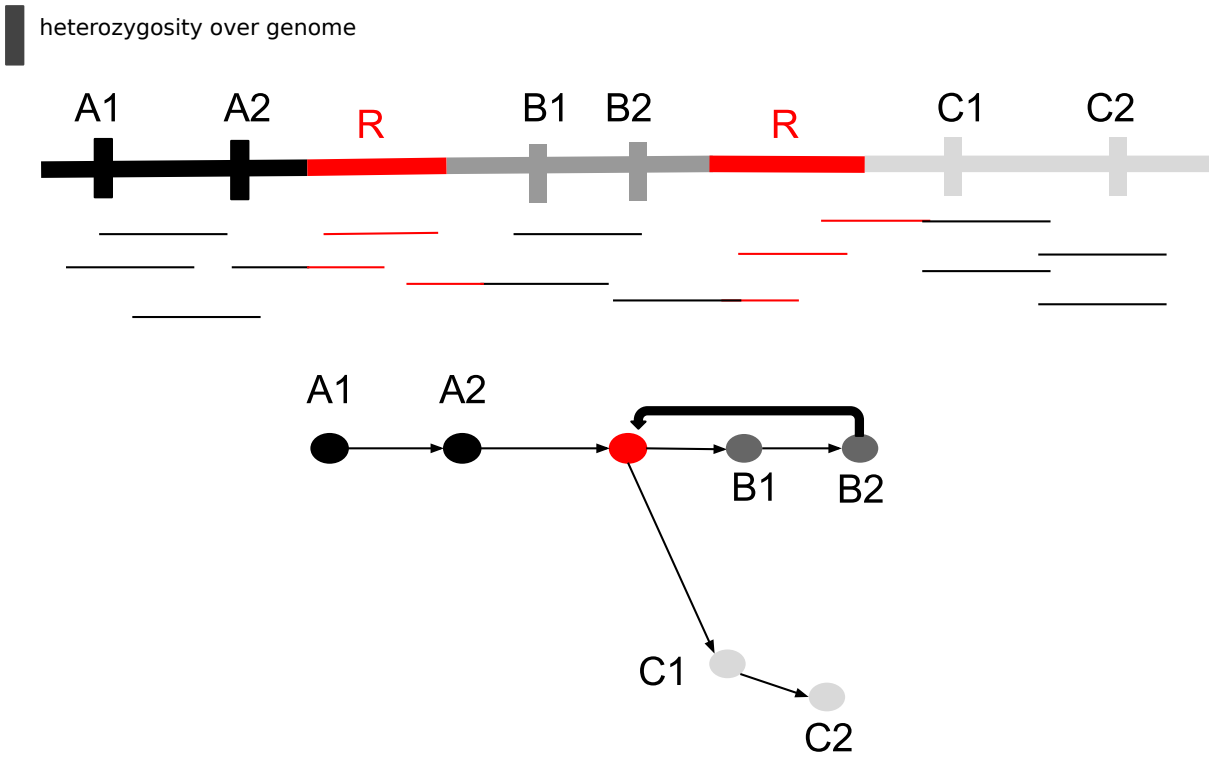

Figure 1.8: The assembly graph in which repetitive and heterozygous regions are condensed as nodes, is shown. At the top, heterozygosity (in vertical bars) and repetitive regions (in red) over the genome are shown. At the bottom, the graph with nodes as heterozygous or repetitive region are shown, and connections are based on the successive read overlap. The graph has cycles because of repetitive region shown by R, which also causes two branches.

NP-hard, when the problem is modeled either as string graphs or de Bruijn graphs (Medvedev et al., 2007). There are several heuristic approaches for approximating the optimal *de novo* haploid assembly based on NGS datasets (Idury and Waterman, 1995; Myers, 1995, 2005; Pevzner et al., 2001; Nagarajan and Pop, 2009, 2013; Sović et al., 2013).

However, even with Sanger (reads of the order of 800-1000 base pairs) and Illumina sequencing, which deliver short reads with low error rates, de novo assembly of heterozygous diploid genomes has been a difficult problem (Vinson et al., 2005; Levy et al., 2007). In practice, there are several short-read assemblers based on Illumina data for heterozygous genomes (Kajitani et al., 2014; Pryszcz and Gabaldón, 2016; Simpson and Durbin, 2012; Bankevich et al., 2012; Li, 2015b). The assemblies that they produce are accurate, but contain gaps and are composed of relatively short contigs and scaffolds. Third generation sequencing technologies such as methods available from Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT) deliver much longer reads, but with high error rates. There are now several long-read assemblers (Koren et al., 2017; Vaser et al., 2017; Xiao et al., 2016; Berlin et al., 2015; Chin et al., 2013; Hunt et al., 2015; Lin et al., 2016) that use these long-read data for de novo assembly. The assemblies that are delivered from these assemblers are more contiguous, with longer contigs and scaffolds. Finally, there are hybrid assemblers that take advantage of long-read data (with its high error rate) and short-read data (with its low error rate) (Bashir et al., 2012; Antipov et al., 2015; Zimin et al., 2017) and attempt to combine the best aspects of both. These hybrid assemblers delivers highly accurate, repeat-resolved assemblies.

The main drawback of the state-of-the-art assemblers mentioned above is that they generate only

one consensus sequence even for diploid organisms. To date, there is only one assembler that can produce diploid assemblies for diploid genomes.

Diploid assembly. A recent and currently only available diploid assembly method — Falcon Unzip (Chin et al., 2016) — is a purely PacBio based diploid assembler. Falcon Unzip generates haplotype contigs or "haplotigs" that represent the diploid genome with correctly phased homologous chromosomes. Falcon Unzip involves constructing a string graph from long PacBio reads, and generating haplotigs in a greedy manner using a local conservative approach.

For generating haplotigs, Falcon Unzip first identifies the phase for each read based on the condition that the read covers at-least one SNV for phasing. In the regions over the genome where heterozygous SNVs are at a long distance from each other, Falcon Unzip can not phase those regions, resulting in incomplete assemblies. Additionally, Falcon Unzip has limitations with respect to phasing all large structural variants and regions with high heterozygosity.

There is no known algorithm that works at different levels of heterozygosity, phases all types of structural variants and generates complete diploid assemblies. A potential approach to achieve the task of complete diploid genomes is to perform phasing directly on the assembly graph. Moreover, it becomes easy to detect large structural variants, such as translocations and other rearrangements, in an assembly graph. Thus, working in the space of assembly graphs provides the opportunity to detect all types of structural variation, which further helps in phasing whole genomes.

Additionally, Falcon Unzip is purely relying on PacBio data, which is noisy and, therefore, it requires high coverage data for producing accurate assemblies. In contrast, hybrid approaches that combine accurate Illumina and long read PacBio data, conceptually have the potential for producing good quality assemblies even at low coverages. However, there is no known algorithm that combines multiple sequencing datasets such as accurate Illumina and long read PacBio data for producing good quality haplotigs.

Open Problem 1.4. Phasing bubbles directly from the assembly graph is an open problem. Additionally, the extension to MEC formulation for phasing reads mapped to assembly graphs do not exist.

1.4 Thesis Scope and Outline

In the above sections, I highlighted four "open problems" in the area of haplotyping using NGS data. The remainder of this thesis is structured as follows:

- Chapter 2 provides a general background on the different types of algorithms. It establishes the motivation on how these algorithms are used in solving small daily examples fast. It highlights the advantages and disadvantages of these algorithms in context of large problems.
- Chapter 3 presents a dynamic programming based algorithm for solving Gapless-MEC instances approximately. It discusses the approximation guarantee, that provides hint about the existence of polynomial time approximation scheme for these instances. (Problem 1.1)
- Chapter 4 discusses different types of NGS datasets, with their advantages and disadvantages. It explores an integrative phasing framework that is obtained for combining NGS datasets. It discusses a parameterized algorithm that solves these instances efficiently in practice. Furthermore, I demonstrate the effectiveness of this algorithm on real genomic datasets. (Problem 1.2)
- Chapter 5 presents a generalized parameterized approach to incorporate information from pedigrees. Furthermore, I show experiments on real datasets and highlight that pedigree data has an additional advantage in delivering better quality haplotypes. (Problem 1.3)
- Chapter 6 focuses on a generalized approach — haplotype-aware diploid assembly — in a graph framework, that has the ability to handle all levels of heterozygosity and structural variations to produce accurate and complete haplotype assemblies. Furthermore, I present this approach as a hybrid of different types of NGS datasets and show its effectiveness on a pseudo-diploid genome. (Problem 1.4)
- Finally, Chapter 7 summarizes the results presented in this thesis, along with an outlook into the future and perspectives.

1.5 Relevant publications

- David Porubsky*, Shilpa Garg*, Ashley D. Sanders*, V. Guryev, Peter M. Lansdorp, T. Marschall, *Dense And Accurate Whole-Chromosome Haplotyping Of Individual Genomes*, Nature Communications, 2017.
- Shilpa Garg, Marcel Martin and Tobias Marschall, *Read-Based Phasing of Related Individuals*, Proceedings of ISMB 2016/Bioinformatics.
- Shilpa Garg, Mikko Rautiainen, Adam M Novak, Erik Garrison, Richard Durbin, Tobias Marschall, *A graph-based approach to diploid genome assembly*, ISMB 2018 (to appear).
- Preprint: Shilpa Garg, Tobias Moemke, *A QPTAS for Gapless-MEC*, Submitted.
- Preprint: M. Martin*, M. Patterson*, Shilpa Garg, S. O. Fischer, N. Pisanti, G. W. Klau, A. Schnhuth, T. Marschall, *WhatsHap: fast and accurate read-based phasing*.
  In this paper, my contribution was in developing some parts of the pipeline and making figures.

Chapter **2**

Algorithmic Background

Many computational problems that arise in practice from analyzing biological data sets are optimization problems. For optimization problems, the goal is to minimize or maximize some objective function (e.g. cost, quality or other measures) over the set of possible solutions that satisfy some constraints. These optimization problems often turn out to be NP-hard and, the NP-hardness of an optimization problem implies that, unless P = NP, there is no polynomial-time algorithm that finds the optimal value of the objective function. The next step would be to look for algorithms that provide near-optimal solutions and work fast in practice. We therefore seek to understand the structure of relevant problem instances and to design algorithmic techniques to exploit these structures.

To design and analyze algorithms for these optimization problems, we explore the theory of parameterized algorithms and the theory of approximation algorithms. Parameterized complexity aims to analyze problem instances in finer detail and some parameters are considered based on the structural and other properties of input or output. The running time of parameterized algorithms is expressed as a function of these parameters. The goal is to identify the parameters for which the overall running time is small when their values are small, even if the input size is large. In approximation algorithms, we relax the optimality criterion: instead of looking for an exact optimal solution, we allow solutions for which the objective function is close to the optimal solution within some worst case bound. We guarantee that the quality of the approximate solution is not worse than that bound on the deviation of the solution from the optimum. The approximation algorithms and the parameterized algorithms can be deterministic or randomized.

2.1 Types of algorithms

In this thesis, we consider the following broadly defined categories of algorithms for solving computational problems: parameterized algorithms, approximation algorithms and randomized algorithms.

2.1.1 Parameterized algorithms

Let us begin with a small example as illustrated by Cygan et al. (2015). Imagine that you are a security guard of a bar in a small town of Germany. On Friday, several people come to the bar for a party. There are some people who often fight with each other at the bar and the guard takes note of them. In order to have a peaceful party without any fights, the guard plans ahead and only admits people who do not fight with anyone else at the bar. At the same time, he is willing to reject at most k people, such that he gains maximum profit.

The above situation can be formalized in the context of an optimization problem. Let us suppose that there are n people who come to the bar, and the constraint is that for each pair of people, we know whether or not they will have a fight between them if the pair is allowed inside the bar. The goal here is to identify the number of people allowed inside the bar to incur maximum profit such that there are at most k people who are prohibited. Figure 2.1 shows an instance of the problem and a solution for k = 3. It can be easily checked that this instance has no solution with k = 2.

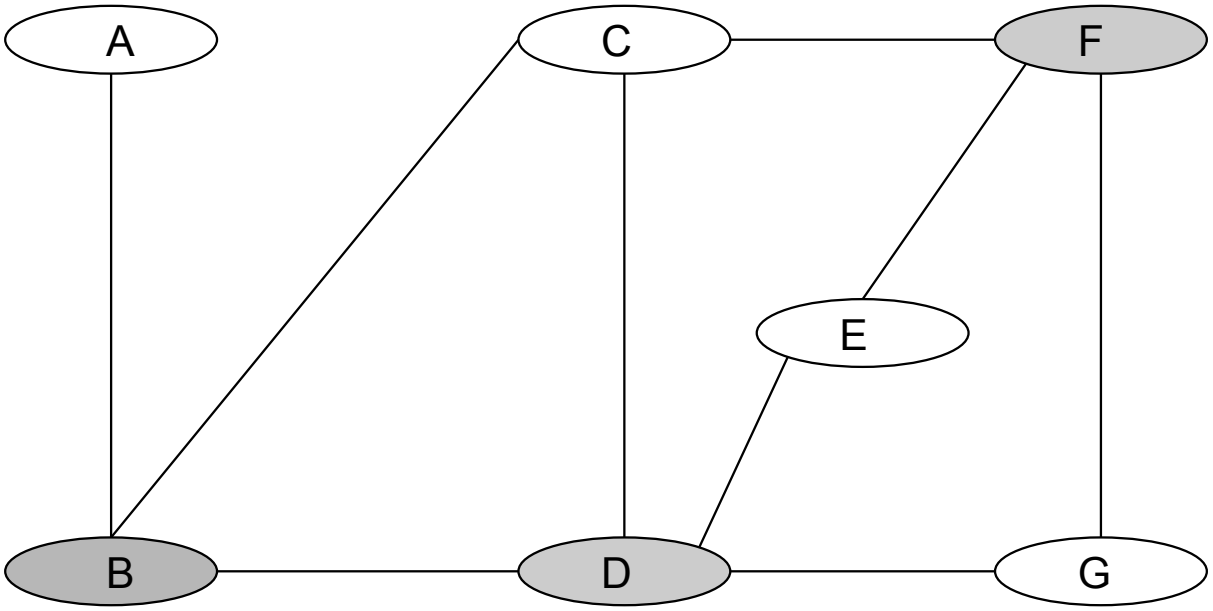

Figure 2.1: An instance of the problem with a solution for k = 3. An edge between two guests means that they will fight if both are admitted. The grey circles represent the troublemakers.

Efficient algorithms. Let us solve the above problem in a naive manner by trying all possibilities. If the problem instance is small, i.e. $n = 100$ people, then the number of possibilities are 2^{100}. Even for such a small instance, the algorithm takes a long time, and cannot give an answer in a reasonable amount of time. Another approach to this problem is to reject a small number of guests, let's say, $k \leq 10$. In this case, the total number of possibilities are $\binom{100}{10}$. This approach is better compared to the brute-force approach, but even this approach takes a long time.

Let us look closer into the problem and, try to identify some peaceful people and troublemakers. In particular, we want to identifying persons who do not have a conflict with anyone, and also identifying a person who fights with at atleast $k + 1$ other persons. The example for this case is shown in Figure 2.1. There are seven persons shown by nodes and there is an edge between nodes if the two persons fight between each other. Here, we want to find a person that fights with at atleast four persons for $k = 3$. In this example, the person whom we want to reject from the party is D, reducing k by one. We proceed ahead in a similar manner. If we are left with no such person then we know that each remaining person will fight with at most k other persons. Thus rejecting any single person resolves at most k potential conflicts. If there are more than k^2 potential conflicts, then it is not possible to have a good party by rejecting only k persons. Taken together, the persons can be classified into two important classes; one, those who participate in at atleast one potential conflict and second, those who participate in at most k conflicts. Thus, in total, there are at most k^2 potential conflicts, which results in $2k^2$ persons for whom it is unknown. Now, if we try all possibilities of $\binom{2k^2}{k}$, the running time is reduced compared to previous approach, but even this approach takes a long time.

After all above insights, the main observation to consider is that every conflict has to be resolved, and the only way to resolve a conflict is to refuse at least one of the two persons. Let us suppose that there is a conflict between persons X and Y. If we add one of them, let's say X, to the list of persons to reject and, then repeat this process by rejecting at most $k - 1$ persons. If it succeeds, we get the solution, otherwise we remove the person X from the reject list and, instead try moving Y to the reject list and run this algorithm recursively. If this succeeds, we get the solution and otherwise, it is not possible to solve this problem by rejecting at most k guests.

To analyze the running time of this recursive algorithm, there are in total $O(2^k)$ recursion calls, 2^k leaves in the recursion tree. Each recursion call can be solved in linear time $O(m + n)$, where m is the

total number of possible conflicts. The good news is that this recursive algorithm gives a solution in a reasonable amount of time. This algorithm runs in time $O(2^k \cdot k \cdot n)$, and is faster than the brute-force algorithm which takes $O(n^k)$ time.

In the $O(2^k \cdot k \cdot n)$— time algorithm, the combinatorial explosion is restricted to parameter k, i.e. the running time is exponential in k, but linear in n. The main insight from the above example is that the problem instances can be solved in time polynomial in the input size, if we fix a parameter. We aim to identify a parameter that is small for the problem instances encountered in practice.

We are now ready to provide some formal definitions on parameterized algorithms as defined by Cygan et al. (2015).

Definition 2.1 (Parameterized algorithms). Algorithms with running time $f(k) \cdot n^c$, for a constant c independent of both n and k, are called *fixed-parameter (FPT) algorithms*.

Typically, the goal in parameterized algorithms is to design FPT algorithms, trying to make both $f(k)$ factors and the constant c in the bound on the running time as small as possible.

In parameterized algorithms, k is simply a *relevant secondary measurement* that encapsulates some aspect of the input instance, be it the size of the solution or the structure and other characterizes of the input instance.

The art of parameterization. We have seen that we can solve a complex problem in an efficient manner by cleverly framing an algorithm using relevant parameters. For some problem instances, it is easy to find such parameters, whereas it is difficult for others.

For example, let us consider a variant of the problem described above where we want to reject at most k persons such that the number of conflicts are at most l. Based on the properties explicitly given in the input instance, the first guess is to parameterize either by k, by l or both. In case of both parameters, the goal is to find an FPT algorithm with running time $f(k, l) \cdot n^c$ for some computable function f depending only on k and l. Thus, the above definition for a single parameter can be extended to considering a set of parameters at the same time.

Parameterized algorithm (more than one parameter): Formally, one can express parameterization by k and l, by defining the value $k + l$ to be the parameter: an $f(k, l) \cdot n^c$ algorithm exists if and only if an $f(k + l) \cdot n^c$ algorithm exists.

Other variants of the problem described above could be formulated as identifying at most k persons to reject such that, say, the numbers of conflicts decreases by p, or such that each allowed person conflicts with at most d other persons, or such that the average number of conflicts per guest is at most a. Based on the properties of this instance, the parameters p, d, a are again explicitly given in the input, indicating the kind of solution to find.

The bottom-line is that we can find different parameters for a problem based on the structure and other special properties of problem instances. This allows us to algorithms which are exponential in these parameters, but polynomial in the input size.

Different problem domains offer different choices of suitable parameters. For example, for the string or sequence problems that are related to genomics, one can parameterize by the maximum read length, by the maximum coverage, by the size of alphabet or by the number of alleles in a variant.

Parameterized complexity allows us to study how different parameters influence the time complexity of the problem. Additionally, it allows us to solve the problem efficiently.

In conclusion, for the same problem, there can be multiple choices of parameters. Selecting the right parameter(s) for a particular problem is often not straightforward.

We follow the formal foundation of parameterized complexity by Cygan et al. (2015).

Definition 2.2. A parameterized problem is a language $L \subseteq \sum^* \times \mathbb{N}$, where $\sum$ is a fixed, finite alphabet. For an instance $(x, k) \in \sum^* \times \mathbb{N}$, k is called the parameter.

Definition 2.3. A parameterized problem $L \subset \sum^* \times \mathbb{N}$ is called *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$ (called a fixed-parameter algorithm), a computable function $f : \mathbb{N} \to \mathbb{N}$ and a constant c such that, given $(x, k) \in \sum^* \times \mathbb{N}$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in

time that is bounded from above by $f(k) \cdot |(x, k)|^c$, where $|(x, k)|$ is the size of an instance. The complexity class containing all fixed-parameter tractable problems is called FPT.

Observe that, given some parameterization problem L, the algorithm designer has essentially two different optimization goals when designing FPT algorithms for L. Since the worst case time complexity has to be of the form of $f(k) \cdot n^c$, one can:

- optimize the *parametric dependence* of the running time, i.e., try to design an algorithm where function f grows as slowly as possible; or
- optimize the *polynomial factor* in the running time, i.e. try to design an algorithm where constant c is as small as possible.

2.1.2 Randomized algorithms

In randomized algorithms, the idea is to perform independent random choices and then utilize these random choices to influence the computation. The assumption is that the average over all random choices results in good performance.

There are two principal advantages to using randomized algorithms. The first is performance — for many problems, randomized algorithms run faster than the best known deterministic algorithms. Thus, randomized techniques are often used in approximation theory in order to provide a faster solution to the problem. Second, many randomized algorithms are simpler to describe and implement than deterministic algorithms of comparable performance.

An observation from basic probability theory —*linearity of expectation*— that is often used in the analysis of randomized algorithms. Linearity of expectation states that the expectation over a sum of random variables is equal to the sum over the expectation of each random variable. For example, for random variables, $X_1, X_2 \ldots$, the linearity of expectation is given by.

$$E\big[\sum_i X_i\big] = \sum_i E[X_i]. \tag{2.1}$$

Here we provide an example of a randomized version of Quick-sort for illustration. Quick-sort is a sorting technique which proceeds as follows. Pick an element p of the array as the pivot. Reorder the elements of the array such that all the elements with values smaller than the pivot are to the left and the greater ones are to the right. After the reordering operation is finished, the pivot is at its final position. The next step is to recursively apply the above steps of randomly picking the pivot and reordering to the left and right parts.

Quick-sort depends on how the pivot element is picked. If the pivot element is chosen randomly, then it is known as randomized Quick-sort. For any given array A of size n, it is easy to show that the expected time of this algorithm is $O(n \log n)$ using the linearity of expectation (Motwani and Raghavan, 2010). Making the algorithm probabilistic gives us more control over the running time. The algorithm runs fast with high probability. They are simple and efficient compared to deterministic algorithms.

2.1.3 Approximation algorithms

In this section, we present some fundamental definitions and concepts of approximation algorithms. Approximation algorithms are a techniques for finding approximate solution for optimization problems that are NP hard. If the approximate solution is close enough to the optimal solution and the algorithm runs faster, then this behavior might be preferable for most practical purposes. Based on this motivation, we define the notion of α-approximation.

We follow the basic definitions of Vazirani (2013).

Definition 2.4. A polynomial-time algorithm A for an optimization problem X is an α-approximation algorithm ($\alpha \geq 1$) if it returns a feasible solution whose value is at most a factor α away from the value of the optimal solution, for any input instance.

Thus, if *opt* is the optimal value of the objective function, an algorithm is an α-approximation if it always returns a solution whose value is at most $\alpha \cdot opt$ for a minimization problem, or at least opt/α for a maximization problem. In general, α can be a growing function of the input size, and does not necessarily need to be a constant number. For some problems, it is possible to find approximate solutions that are arbitrarily close to the optimal solution. More formally:

Definition 2.5. We say that an algorithm A is a polynomial time approximation scheme (PTAS) for an optimization problem X if for every fixed $\varepsilon > 0$ and for any instance I, the running time of $A(\varepsilon, I)$ is polynomial in the input size n, and it returns a solution whose value is at most a $1 + \varepsilon$ factor away from the value of the optimal solution.

Definitions 2.4 and 2.5 can be generalized in the context of randomized algorithms. In particular, we call an algorithm an expected α-approximation if the expected value of the output solution satisfies the above constraints.

For a PTAS, the running time is polynomial for a fixed ε, but the dependency on ε can be arbitrary; in fact, running times can be on the order of $f(1/\varepsilon)n^{O(g(1/\varepsilon))}$ for some functions f and g that are super-polynomial with respect to $1/\varepsilon$. Based on these functions f and g, there are different variants of PTAS algorithms. If the function g is a constant that does not depend on ε, the algorithm is called Efficient PTAS (EPTAS); if, moreover, f is polynomial in $1/\varepsilon$, then it is called a Fully Polynomial Time Approximation Scheme (FPTAS). In some sense, NP-hard problems admitting an FPTAS can be thought as the easiest hard problems; one such example is the Knapsack Problem.

A relaxation of Definition 2.5 that allows for a slightly larger running time is a Quasi-Polynomial Time Approximation Scheme (QPTAS). A QPTAS is defined exactly same as above, except that A is allowed quasi-polynomial time $O(n^{poly \log n})$.

The class of problems that admit a constant factor approximation in polynomial time is called APX. Clearly, all problems that admit a PTAS are in APX, but the converse is not true if $P \neq NP$ (Jansen, 1998).

Some problems are known to be APX-hard. If a problem is APX-hard, then the existence of a PTAS for it would imply the existence of a PTAS for every problem in APX. Thus, being APX-hard is considered a strong evidence that the problem does not admit a PTAS.

The existence of a QPTAS for a problem is sometimes seen as a hint that a PTAS might exist: in fact, it implies that the problem is not APX-hard.

For some problems, we can obtain better approximation ratios if we assume that the solution is large enough. Formally, for a minimization problem X, the asymptotic approximation ratio ρ of an algorithm A is defined as:

$$\rho = lim_{n \to \infty} sup_I \{apx(I)/opt(I) : opt(I) \geq n\}$$

where $apx(I)$ and $opt(I)$ are the objective function value computed by algorithm A and optimal value solution on the instance I respectively. Similarly to the definition of PTAS, we say that an algorithm A is an Asymptotic PTAS or APTAS for problem X if $A(\varepsilon, .)$ is an asymptotic $(1 + \varepsilon)$-approximation for any fixed $\varepsilon > 0$.

There are several advantages to using approximation algorithms.

- An approximation algorithm provides a way to find a near-optimal solutions when the optimal solution is not required, and finding an optimal solution for the problem is NP hard.
- An approximation algorithm provides a mathematically rigorous basis to study heuristics, and helps in unraveling structures of problem instances.
- The field of approximation algorithms gives us a means of distinguishing between various optimization problems in terms of how well they can be approximated.

Let us consider a classic example to illustrate the motivation and design of an approximation algorithm. Imagine that a thief has entered a house when its inhabitants have gone on a vacation. The thief has a knapsack which is of fixed weight. In the house, the thief has multiple options for valuable items to steal. Each of these items has a fixed price and weight as shown in Table 2.1. Since the knapsack that can carry a fixed weight of 12, the thief can not steal all the items. The thief's goal is to steal the items

such that he gains the maximum profit from the items that fit into his knapsack. The thief problem actually is the popular Knapsack Problem.

Efficient algorithms. The naive approach that a thief thinks of, is to try all possibilities of items and then choose the best. The total number of possible combinations are 2^n, which takes long time to compute the final solution.

The next approach the thief follows is sorting the items in non-decreasing order based on the price. The thief would first include an item in the knapsack with the maximum price and so on. In this example, the thief first takes the gold ring and then, the mobile phone, which results in a weight of 11 and a price of 30. This approach is basically a greedy approach and does not always give the optimal solution. For instance, in this example, the optimal solution consists of items 1, 3 and 4, for a total weight of 31.

Another technique that the thief can employ is dynamic programming. The thief considers sub-problems based on different items. The thief stores the information "the best knapsack so far" for these sub-problems in a DP table. The main idea is that the thief does not have to re-compute some sub-problems, which helps to save time.

Running time. The run-time of this approach depends on the size of the DP table. Its two factors are k rows (determined by the target capacity), and the n columns (the number of items in the set). So, the run-time is $\theta(k\dot{n})$. The main point here is that k is not a constant, and its size may be considerably larger than the number of items n in the set. Therefore, it is inefficient to solve this problem using a deterministic algorithm. However, these instances often arise in practice and we would like to solve them. Thus we want to look into its approximate solution such that the algorithm to generate this solution is faster than a deterministic algorithm.

We now provide its formal definition and its approximate solution.

In the knapsack problem, given a knapsack of size $B \in Z^+$ and a set $S = \{a_1, ..., a_n\}$ of items with their corresponding sizes and profits $s(a_i) \in Z^+$ and $p(a_i) \in Z^+$, the goal is to find an optimal subset of objects whose total size is bounded by B and which yields the maximum possible total profit. This problem is also called as the 0/1 knapsack problem because each item can either be included in or excluded from the knapsack.

PTAS for Knapsack. We present an algorithm, which was proposed by Lawler (1979), where it is assumed that we know the optimal solution OPT. Let us assume that the set $O \subset S$ is considered by the optimal solution. Another set $Q \subset O$ contains the items with value $\geq \varepsilon \cdot OPT$. We try all possibilities for sets Q such that $|Q| \leq 1/\varepsilon$. For some Q we remove an item from a set with value $\geq \varepsilon \cdot OPT$ and then run greedy algorithm to obtain set G. The final solution is the best value obtained by the union of Q and G. The running time of this algorithm is $(n+1)^{1/\varepsilon} \cdot \text{poly}(n)$.

For analyzing this algorithm, let us consider a case when Q is guessed correctly, then the loss from greedy algorithm is at most $\varepsilon \cdot OPT$. This happens due to the bound on the size on items that are removed, which are bounded by $\varepsilon \cdot OPT$. Therefore, it is easy to see that $p(Q \cup G) = OPT - \varepsilon \cdot OPT = (1-\varepsilon)OPT$.

The above analysis shows that the Knapsack problem is in PTAS. In addition, the problem exhibits full polynomial time approximation scheme (FPTAS).

Item	weight	price
gold ring	3	20
mobile	8	10
radio	4	8
shoes	5	3

Table 2.1: Example for Knapsack problem

Chapter **3**

Approximation algorithm for phasing individual genomes

In this chapter, we study the approximation status of Gapless-MEC, which is a variant of MEC problem (see Problem 1.1 introduced in Chapter 1). In other words, the minimum error correction problem (MEC) is closely related to segmentation problems that were introduced by [Kleinberg–Papadimitriou–Raghavan STOC'98] in the context of data mining. A MEC instance is an $n \times m$ matrix M with entries from $\{0, 1, -\}$. Feasible solutions are composed of two binary m-bit strings, together with an assignment of each row of M to one of the two strings. The objective is to minimize the number of mismatches (errors) where the row has a value that differs from the assigned solution string. The symbol "–" is a wildcard that matches both 0 and 1. A MEC instance is *gapless*, if in each row of M all binary entries are consecutive.

Without restrictions, it is known to be UG-hard (Trevisan, 2012) to compute an $O(1)$-approximate solution to MEC. For both MEC and Gapless-MEC, the best polynomial time approximation algorithm has a logarithmic performance guarantee (Bonizzoni et al., 2016). We partially settle the approximation status of Gapless-MEC by providing a quasi-polynomial time approximation scheme (QPTAS). Additionally, for the relevant case where the binary part of a row is not contained in the binary part of another row, we provide a polynomial time approximation scheme (PTAS).

3.1 Our results.

Our main result is the following theorem.

Theorem 3.1. There is a quasi-polynomial time approximation scheme (QPTAS) for Gapless-MEC.

We therefore partially settle the approximability for Gapless-MEC, which is not APX-hard unless NP $\subseteq$ QP (cf. Remy and Steger (2009)). Moreover, Cilibrasi et al. (2007) showed that allowing a single gap in each string renders the problem APX-hard. Thus our result reveals a separation of the hardness of the gapless case and the case where we allow a single gap. Furthermore, already Binary-MEC is strongly NP-hard since the input does not contain numerical values. Therefore we can exclude the existence of an FPTAS for both Binary-MEC and Gapless-MEC unless P = NP.

Additionally, we address the class of *subinterval-free* Gapless-MEC instances where no string is contained in another string. More precisely, for each pair of rows from M we exclude that the set columns with binary entries from one row is a strict subset of the set of columns with binary entries from the other row.

Theorem 3.2. There is a polynomial time approximation scheme (PTAS) for Gapless-MEC restricted to instances such that no string is the substring of another string.

3.2 Further related work.

Binary-MEC is a variant of the Hamming k-Median Clustering Problem when $k = 2$ and there are PTAS known (Jiao et al., 2004; Ostrovsky and Rabani, 2002). Li et al. (2002) provided a PTAS for the general consensus pattern problem which is closely related to MEC. Additionally, they provided a PTAS for a restricted version of the star alignment problem aligning with at most a constant number of gaps in each sequence. More recently, Bonizzoni et al. (2016) showed that it is unique games hard to approximate MEC with constant performance guarantee, whereas it is approximable within a logarithmic factor in the size of the input. Gapless-MEC was shown to be NP-hard by Cilibrasi et al. (2007).[1]

Alon and Sudakov (1999) provided a PTAS for H2S, the maximization version of Binary-MEC and Wulff et al. (2013) showed that there is also a PTAS for the maximization version of MEC. For Gapless-MEC, He et al. (2010) studied the fixed-parameter tractability in the parameter of fragment length with some restrictions. These restrictions allow their dynamic programming algorithm to focus on the reconstruction of a single haplotype and, hence, to limit the possible combinations for each column. There is an FPT algorithm parameterized by the coverage (Patterson et al., 2015). Bonizzoni et al. (2016) provided FPT algorithms parameterized by the fragment length and the total number of corrections for Gapless-MEC.

Most research in haplotype phasing deals with exact and heuristic approaches to solve MEC. Exact approaches, which solve the problem optimally, include integer linear programming by Fouilhoux and Mahjoub (2012) and fixed-parameter tractable algorithms by He et al. (2010); Pirola et al. (2015). There is a greedy heuristic approach proposed to solve Binary-MEC (Bansal and Bafna, 2008).

Lancia et al. (2001) obtained a network-flow based polynomial time algorithm for Minimum Fragment Removal (MFR) for gapless fragments. Additionally, they found the relation of Minimum SNPs Removal (MSR) to finding the largest independent set in a weakly triangulated graph.

3.3 Overview of our approach.

Our algorithm is a dynamic program (DP) that is composed of several levels. Given a general Gapless-MEC instance, we decompose the rows of the instance into length classes according to the length of the contiguous binary parts of the rows as shown in Figure 3.1.

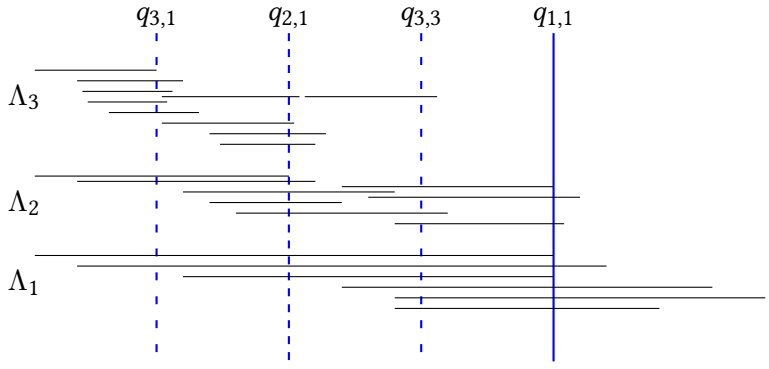

Figure 3.1: Different length classes, Λ_1 with corresponding column $q_{1,1}$, Λ_2 with corresponding columns $q_{2,1}$, $q_{2,2} = q_{1,1}$, and Λ_3 with corresponding columns $q_{3,1}$, $q_{3,2} = q_{2,1}$, $q_{3,3}$, $q_{3,4} = q_{1,1}$.

For each length class we consider a well-selected set of columns such that each row crosses at least one column and at most two. (Row i crosses a column j, if $M_{i,j} \in \{0, 1\}$, i.e., the binary part of the row contains that column.)

We further decompose each length class into two sub-classes, one that crosses exactly one column and one that crosses exactly two columns as shown in Figure 3.2.

[1] Their result predates the hardness result of Feige (2014) for H2S. The proof of the claimed NP-hardness of H2S by Kleinberg et al. (1998) was never published.

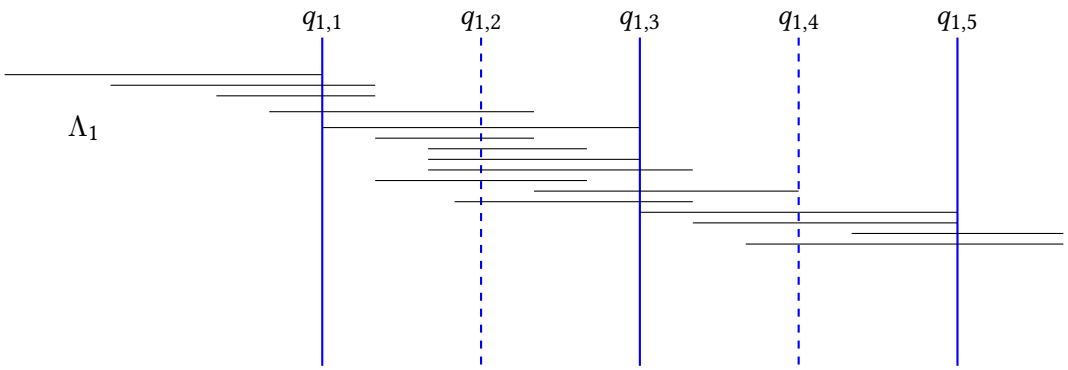

Figure 3.2: For a single-length-class instance, the sketch shows the strings crossing each column either exactly once or exactly twice.

For the second class, it is sufficient to consider every other column, which leaves us with many *rooted* instances. Thus for each sub-instance there is a single column (the root) which is crossed by all rows of the instance.

We further decompose rooted sub-instances into the left hand side and the right hand side of the root. Since the two sides are symmetric, we can arrange the rows and columns of these sub-instances in such a way that all rows cross the first column. We call this type of sub-instance *SWC-instance* (for "simple wildcards") as shown in Figure 3.3. We order the rows from top to bottom by increasing length in order to be able to further decompose the instance.

Figure 3.3: Simple Wildcard (SWC) instances

The first level of our DP solves these highly structured SWC-instances. The basic idea that we would like to apply is that we select a constant number of rows from the instance that represents the solution. Without further precautions, however, this strategy fails because of differing densities within the instance: the selected rows have to represent both the entries of columns crossed by many short rows and entries of arbitrarily small numbers of rows crossing many columns. To resolve this issue, we observe that computing the solution strings σ and σ' is equivalent to finding a partition of M into two row sets, one assigned to σ and the other assigned to σ'. If we assume to have the guarantee that for both solution strings σ and σ' an ε fraction of rows of the matrix M forms a Binary-MEC sub-instance, we show that the basic idea works.

This insight motivates to separate SWC-instances from left to right into sub-instances with the required property and to assemble them from left to right using a DP. There are, however, several complications. In order to choose the right sub-instances, we have to take into account that the choice depends on which rows are assigned to σ and which are assigned to σ'. Therefore the DP has to take special care when identifying the sub-instances.

Furthermore, in order to stitch sub-instances together to form a common solution, the solution computed in the left sub-instance has to compute a set of candidate solutions oblivious of the choices of

the right sub-instance. This means that we have to compute a solution to the left sub-instance without looking at a fraction of rows. We present an algorithm for these sub-instances in Section 3.5.

In order to combine the sub-instances, we face further technical complications due to having distinct sub-instances for those rows assigned to σ and those rows assigned to σ'. In Section 3.5.1, we introduce a DP whose DP cells are pairs of simpler DP cells, one for σ and one for σ'.

Before we consider general instances, we first develop our techniques by considering subinterval-free instances which are easier to handle (Section 3.6). Observe that the instances considered until now are special rooted sub-interval-free instances. We show how to solve arbitrary rooted sub-interval-free instances by combining the DP with additional information about the sub-problems that contain the root. We then introduce the notion of domination in order to combine rooted sub-interval-free instances with a DP proceeding from left to right. The main idea is that a dominant sub-problem dictates the solution. At the interface of two sub-instances, there can be a (contiguous) region where none of the two sub-problems is dominant. We show that these regions can be solved directly by considering a constant number of rows (using the results from Section 3.5).

Until this point, all parts of our algorithm run in polynomial time. We lose this property when considering length classes, in Section 3.7.1. The length classes allow us to separate an instance into rooted sub-instances. The difficulty is that the left hand side of a separating column may have a completely different structure than the right hand side of that column. We do not know how to combining the two sides by considering only a polynomial number of possibilities. If we allow, however, quasipolynomial running time, we can solve the problem. We use that each of the two sub-instances (the one on the left and the one on the right) is composed of at most logarithmically many parts. Considering all parts simultaneously allows us to take care of dependencies between the left hand side and the right hand side and still solve them as if they were separate instances.

Combining such rooted instances from left to right then can be done in the same spirit as combining rooted sub-interval-free instances. To solve the entire length-class, we combine both solutions by running a new DP that considers quadruples of DP cells.

Finally, in Section 3.7.2, we are able to handle all length classes simultaneously. We solve general instances in the same spirit as the combined sub-instances of a single length class. Instead of considering quadruples of cells, however, we form collections of quadruples that are – figuratively speaking – stacked on top of each other. The key insight is that there are only $O(\log(n))$ different length classes and each collection has at most one quadruple of each length class. Considering all possible collections adds another power of $\log(n)$ to the running time, which is still quasi-polynomial.

3.4 Preliminaries and notation.

We consider a Gapless-MEC instance, which is a matrix $M \in \{0,1,-\}^{n\times m}$. The ith row of M is the vector $M_{i,*} \in \{0,1,-\}^{1\times m}$ and the jth column is the vector $M_{*,j} \in \{0,1,-\}^{n\times 1}$. The length of the binary part in $M_{i,*}$ is $|M_{i,*}|$. We say that the ith row of M *crosses* the jth column if $M_{i,j} \in \{0,1\}$.

For each feasible solution (σ,σ') for M, we specify an assignment of rows $M_{i,*}$ to solution strings. The default assignment is specified as follows. For a row $M_{i,*}$, we assign $M_{i,*}$ to σ if $\mathrm{dist}(\sigma, M_{i,*}) \leq \mathrm{dist}(\sigma', M_{i,*})$. Otherwise we assign $M_{i,*}$ to σ'. For the rows of M assigned to σ we write $\sigma(M)$ and for the rows assigned to σ' we write $\sigma'(M)$. For a given instance, Opt $= (\tau,\tau')$ denotes an optimal solution. Observe that knowing Opt allows us to obtain an optimal assignments $\tau(M)$ and $\tau'(M)$ by assigning each row to the solution string with fewest errors and knowing $\tau(M)$ and $\tau'(M)$ allows us to obtain an optimal solution by selecting the column-wise majority values.

3.5 Simple instances with wildcards.

In this section, we consider instances of Gapless-MEC where all entries of column one in M are zero or one, i.e., $M_{i,1} \in \{0,1\}$ for each index i. Observe that the wildcards now have a simple structure which we refer to as SWC-structure. An instance with SWC-structure is an SWC-instance.

Definition 3.1 (Standard ordering of SWC-instances). We define the *standard ordering* of rows in M such that $|M_{i,*}| \leq |M_{i+1,*}|$ for each i, i.e., we order them from top to bottom in increasing length of the binary part.

Definition 3.2 (Good SWC-instances). We call an SWC-instance M *good*, if it is in standard ordering and there are at least $\varepsilon|\tau(M)|$ rows of $\tau(M)$ and at least $\varepsilon|\tau'(M)|$ rows of $\tau'(M)$ that have only entries from $\{0, 1\}$.

To solve good SWC-instances, we generalize the PTAS for Binary-MEC by Jiao et al. Jiao et al. (2004). Our algorithm requires partitions of the set of rows. In the following two definitions, the required number of rows may be a fractional number. To solve the problem, we allow the assignment of fractional rows, i.e., for a row i, we can choose an $x \in [0, 1]$ and assign an x fraction of i to one set and a $1 - x$ fraction to the other set.

The following two definitions allow us introduce a structured view on optimal solutions.

Definition 3.3 (Trisection). An *ε-trisection* of an instance M for τ is a partition of the rows into three consecutive ranges that have the following properties.

1. The first range U contains row $M_{1,*}$ and $(1 - \varepsilon)|\tau(M)|$ rows of $\tau(M)$.
2. The second range L is consecutive to first row set containing $(\varepsilon - \varepsilon^2)|\tau(M)|$ rows of $\tau(M)$.
3. The third range X contains the remaining rows in M.

To avoid ambiguity, we choose L and X such that the first row is in $\tau(M)$.

We define an ε-trisection U', L', and X' for τ' analogously, replacing $\tau(M)$ by $\tau'(M)$.

Definition 3.4 (Subdivision of trisections). We consider the rows sets U, L, U', L' from Definition 3.3 and additionally, we divide each of these sets into $1/\varepsilon^2$ disjoint subsets denoted as U_i, L_i, U'_i, L'_i. For each i, U_i contains $\varepsilon^2 \cdot |U|$ rows from $\tau(M)$ and L_i contains $\varepsilon^2 \cdot |L|$ rows from $\tau(M)$. Analogously, each U'_i contains $\varepsilon^2 \cdot |U'|$ rows from $\tau'(M)$ and L'_i contains $\varepsilon^2 \cdot |L'|$ rows from $\tau'(M)$. To avoid ambiguity, each set U_i and L_i starts with a (fractional) row of $\tau(M)$ and each set U'_i and L'_i starts with a (fractional) row of $\tau'(M)$.

We introduce a new algorithm SWC_δ for our setting. For an instance M, we consider the rows sets U, L, U', L' from the ε-trisections of M and their subsets according to Definition 3.4. Additionally, we select a multi-set of rows from $U'_i \cap \tau'(M)$ and $L'_i \cap \tau'(M)$. We then compute the majority weighting according to Definition 3.5 for each column j using multisets based on the minimum number of errors. The main idea to find two small row sets that represent the whole instance M. The intuitive meaning is that we select rows from the upper part with a much lower density then the rows of the lower part. We therefore introduce a bias such that all rows are equally important.

Definition 3.5 (Weighted majority). Let j be an integer and let $\tilde{U}$ and $\tilde{L}$ be two matrices with at least j columns. In $\tilde{U}_{*,j}$ and $\tilde{L}_{*,j}$, we replace all zeros by -1 and then all wildcard symbols by zero. We then compute the number $\nu := \sum_{i \in \tilde{U}_{i,j}} (1 - \varepsilon)i/(\varepsilon - \varepsilon^2) + \sum_{i \in \tilde{L}_{i,j}} i$. Then $\textsc{Majority}_j(\tilde{U}, \tilde{L}) = 0$ if $\nu < 0$ and $\textsc{Majority}_j(\tilde{U}, \tilde{L}) = 1$ if $\nu \geq 0$.

With this preparation, we are now ready to present the algorithm. The input has a long list of parameters that will allow our dynamic programs later on to control the execution. The reason is that we do not know τ and τ'. Therefore the algorithm takes *guesses* of row sets as input. The values r and r' are guesses of $|\tau(M)|$ and $|\tau'(M)|$.

Input : Row sets U_i, L_i, U'_i and L'_i of a good SWC-instance M, numbers r, r'.
Optional: selection of rows $\tilde{U}_i, \tilde{L}_i, \tilde{U}'_i, \tilde{L}'_i$, see below.
Output: A pair of solution strings (σ, σ').

1 Run the algorithm for each possible selection of the following type and keep the best outcome (minimum number of errors); `// If provided as input, skip selection.`
2 For each i, select (with repetition) a multi-set $\tilde{U}_i$ of $1/\delta$ rows from U_i and $\tilde{L}_i$ from L_i;
3 For each i, select (with repetition) a multi-set $\tilde{U}'_i$ of $1/\delta$ rows from U'_i and $\tilde{L}'_i$ from L'_i such that $\tilde{U}' \cap \tilde{U} = \tilde{L}' \cap \tilde{L} = \emptyset$;
// $\tilde{U} := \bigcup_i \tilde{U}_i$. `The values` $\tilde{U}'$, $\tilde{L}$, `and` $\tilde{L}'$ `are defined analogously.`
4 For each column j, set $\sigma_j := \textsc{Majority}_j(\tilde{U}, \tilde{L})$ and $\sigma'_j := \textsc{Majority}_j(\tilde{U}', \tilde{L}')$;
5 For each row i of M, determine the value $d_i := \text{dist}(\sigma, M_{i,*}) - \text{dist}(\sigma', M_{i,*})$;
6 Assign the r rows with minimal values d_i to σ and the remaining r' rows to σ'.

Algorithm 1: SWC_δ

Observe that for small (i.e., constant) values of r or r', the algorithm SWC_δ can be replaced by an exact algorithm since we know $\tau(M)$ if and only if we know $\tau'(M)$, and we are able to guess constantly many rows.

Lemma 3.1. Let M be a good SWC-instance. For sufficiently large $r = |\tau(M)|$ and $r' = |\tau'(M)|$, let U_i, L_i, U'_i, L'_i be a subdivision (Definition 3.4) of an ε-trisection U, L, X, U', L', X' of M. Then $\text{SWC}_{\varepsilon^3}$ is a $(1 + O(\varepsilon))$-approximation algorithm for M.

The proof is based on a randomized argument using Chernoff bounds. (See Appendix A.1).

Lemma 3.1 shows that the set of solutions considered by $\text{SWC}_{\varepsilon^3}$ contains at least one solution that is good enough even though we do not look at X. It does not say that we finally compute that solution, since other solutions may have fewer errors in $U \cup L$ or $U' \cup L'$. For our dynamic programs, we need a stronger statement. We would like to be able to compute a solution for an instance and *afterwards* change a fraction of assignments without losing the approximation guarantee. The next lemma is a key ingredient of our result.

Lemma 3.2. Let M be a good SWC-instance and $\varepsilon > 0$ sufficiently small. Let U, L, X be an ε-trisection for $\tau(M)$ and U', L', X' an ε-trisection for τ', with subdivisions U_i, L_i, U'_i, L'_i according to Definition 3.4. Let (σ, σ') be the solution computed by $\text{SWC}_{\varepsilon^3}$ with $r = |\tau(M)|$, $r' = |\tau'(M)|$. Then re-assigning the rows $\sigma(X)$ to $\tau(X)$ and $\sigma'(X')$ to $\tau'(X')$ gives a $(1 + O(\varepsilon))$-approximation for the instance M.

Proof. For ease of presentation, we assume that all appearing numbers are integers. It is easy to adapt the proof by rounding fractional numbers appropriately.

We first analyze the computed solution string σ. Let η be the total number of errors of (τ, τ') within M and let η_P be the total number of errors of (σ, σ') within $P := U \cup L$. Due to Lemma 3.1, we have $\eta_P \leq (1 + O(\varepsilon))\eta$.

We may assume $r \geq r'$ since otherwise we can simply rename the two strings τ, τ'. Additionally, by renaming of σ and σ', we may assume that $|\sigma(P) \cap \tau(P)| \geq |\sigma'(P) \cap \tau(P)|$. Therefore $|\tau(P)| \geq n/3$ and $|\sigma(P) \cap \tau(P)| \geq n/6$. (Recall that the matrix M has n rows and m columns. The value $n/3$ is a save bound on $n/2 - \varepsilon^2 n$.)

Claim 3.1. There is a set I of $m - 25\eta/n$ indices j such that $\sigma_j = \tau_j$ for all $j \in I$.

Proof of Claim. We concentrate on the columns of M where both strings τ and σ have at most $n/12$ errors within P. By counting the errors, there are at most $12\eta/n$ columns where τ has at least $n/12$ errors. Similarly, there are at most $12(1 + O(\varepsilon))\eta_P/n < 13\eta/n$ many columns where σ has at least $n/12$ errors. Therefore there is a set I of at least $m - 25\eta/n$ columns where simultaneously both τ and σ have less than $n/12$ errors each.

Now suppose that the claim was not true and there was an index $j \in I$ with $\tau_j \neq \sigma_j$. Then, since $|\tau(P) \cap \sigma(P)| \geq n/6$, either σ_j or τ_j is erroneous in at least $n/12$ rows of $\tau(P) \cap \sigma(P)$, a contradiction. ◊

Next we analyze σ' for the columns I. Let j be a column (i.e., an index) from I. By symmetry, we may assume $\sigma_j = \tau_j = 0$. We aim to show that an optimal solution has always sufficiently many errors to pay for wrong entries of σ'.

Let η_j be the number of errors of (τ, τ') in column j of M and let $\eta_{P,j}$ be the number of errors of (σ, σ') in column j of P. Let $\eta''_j = \eta_j + \eta_{P,j}$.

Claim 3.2. For each column j of I, either $\sigma'_j = \tau'_j$ or $\eta''_j \geq (\varepsilon - \varepsilon^2)|\tau'(M)|/2$.

Proof of Claim. We distinguish two cases. We first assume $\tau'_j = 0$. If also $\sigma'_j = 0$, we are done. We therefore assume $\sigma'_j = 1$. If there are more than $|\tau'(L')|/2$ ones in column j of L', (τ, τ') has more than $|\tau'(L')|/2$ errors in column j and thus $\eta_j \geq |\tau'(L')|/2$. Otherwise $\sigma'(L')$ has at least $|\tau'(L')|/2$ zeros in column j and therefore $\eta_{P,j} \geq |\tau'(L')|/2$. We obtain $\eta''_j \geq |\tau'(L')|/2 \geq (\varepsilon - \varepsilon^2)|\tau'(M)|/2$ as claimed.

In the second case, $\tau'_j = 1$ and we assume that $\sigma'_j = 0$. If there are more than $r'/2$ ones in column j of U', (σ, σ') has more than $r'/2$ errors in column j and thus $\eta_{P,j} \geq |\tau'(U')|/2$. Otherwise $\tau'(U')$ has at least $r'/2$ zeros in column j and therefore $\eta_j \geq |\tau'(U')|/2$. Again, we obtain $\eta''_j \geq |\tau'(U')|/2 \geq (1 - \varepsilon)|\tau'(M)|/2$ as claimed. ♢

Since by our assumption $|\tau'(X')| < \varepsilon^2|\tau'(M)|$, Claim 3.2 implies that within I, after reassigning the rows we still have a $(1 + O(\varepsilon))$-approximation.

To finish the proof, we argue that η is large enough to pay for all errors in X and X' outside of I. Let η_I be the number of errors due to assigning σ to $\tau(X)$ and σ' to $\tau'(X')$ within the interval I.

Then, using the size of I stated in Claim 3.1, the total number of errors of (σ, σ') in M is at most $(1 + O(\varepsilon))\eta + \eta_I + \varepsilon^2 n \cdot 25\eta/n$, i.e., the errors of $\mathrm{SWC}_{\varepsilon^3}$ within P, the errors within X and X' in the columns of I, and all other entries of $X \cup X'$. The obtained approximation ratio is

$((1 + O(\varepsilon))\eta + \eta_I + \varepsilon^2 n \cdot 25\eta/n/\eta \leq (\eta + O(\varepsilon)\eta + 25\varepsilon^2\eta)/\eta = 1 + O(\varepsilon)$.

The first inequality uses that for some constant k, $(1 + k\varepsilon)\eta \geq \eta + \eta_I$. □

3.5.1 A DP for SWC-instances.

Let M be an SWC-instance with rows $\{1, 2, \ldots n\}$. We define i to be the start and end_i the end of string number i of M, i.e., the column number of the matrix where the binary part starts and ends. For a sub-matrix M' of M, M' determines the index of the first column of M' and $\mathrm{end}_{M'}$ the index of the last column of M'.

We next specify the parts of which the DP cells are composed. We divide the input instance into blocks defined as follows.

Definition 3.6 (Block). Given a good SWC-instance M, a block B is a sub-instance determined by three numbers $1 \leq a < b < c \leq n$ as follows. The first column of B is column 1 of M. The last column of B is end_b. The first row of B is a and the last row is n. We write U_B for the rows from a to $b - 1$, L_B for the rows from b to $c - 1$, and X_B for the rows from c to n.

The idea is that a block determines a trisection. We subdivide each block into chunks and select rows from these chunks. Chunks are closely related to subdivisions of trisections, but we do not assume the knowledge of (τ, τ').

Definition 3.7 (Chunk). Let B be a block determined by the numbers a, b, c. We partition B into $2/\varepsilon^2$ many *chunks* (ranges or rows). These chunks are determined by numbers

$$a = a_1 < a_2 < \cdots < a_{1/\varepsilon^2+1} = b = b_1 < b_2 < \cdots < b_{1/\varepsilon^2+1} = c\,.$$

The ℓth chunk of U_B is the submatrix composed of the rows a_ℓ to $a_{\ell+1} - 1$ and the ℓth chunk of L_B is the submatrix composed of the rows b_ℓ to $b_{\ell+1}$.

Definition 3.8 (Selection). For each block B with a set of chunks C, we consider multiset T of rows of size $2/\varepsilon^5$. We require that T contains $1/\varepsilon^3$ rows from each chunk in C.

The selection T will take the role of $\tilde{U}$ and $\tilde{L}$ in SWC_δ. With these preparations we can define a DP cell.

Definition 3.9 (DP cell). For each block B, each set of chunks C of B and each selection T of rows from B, there is a DP cell represented by $D(B, C, T)$. A DP cell $D(B, C, T)$ is a *predecessor* of $D(\hat{B}, \hat{C}, \hat{T})$ if the following conditions hold.

- $\hat{a} = b$ and $\hat{b} = c$, where $b, c, \hat{a}, \hat{b}$ are the numbers from Definition 3.6.
- The chunks from C between b and c are exactly the chunks from $\hat{C}$ between $\hat{a}$ to $\hat{b}$.
- For each pair of chunks from $T \times \hat{T}$ with the same range of rows, the selection T matches the selection $\hat{T}$.

The value of $D(B, C, T)$ will be an approximation of the minimum number of errors that we can have in M until the last column of B.

We now describe the dynamic program for a pair of solution strings (σ, σ') by using joint DP cells (ζ, ζ') (see also Fig. 3.4).

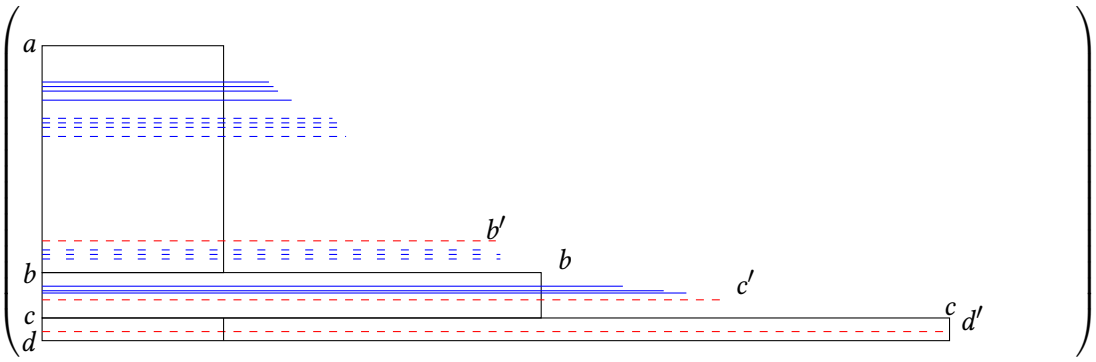

Figure 3.4: Example for a pair of strings with $b' > b$. The blue lines and dashed blue ones represent sets T and T', and $T \cap T' = \emptyset$.

For σ', we use the same notation as in Definitions 3.6, 3.7 and 3.8, but we use the symbol prime ($\cdot'$) for all occurring variables.

Definition 3.10 (DP cell for a pair). We define joint DP cell $(\zeta, \zeta') = (D(B, C, T), D'(B', C', T'))$ with the two single cells defined as in Definition 3.9. We require that

- the rows of C and C' where chunks start are pairwise distinct, and
- $T \cap T' = \emptyset$.

Definition 3.11 (Predecessor of a joint DP cell). A DP cell $(\hat{\zeta}, \hat{\zeta}')$ is a *predecessor* of (ζ, ζ') if (i) $\hat{\zeta} = \zeta$ and $\hat{\zeta}'$ is a predecessor of ζ'; or (ii) $\hat{\zeta}$ is a predecessor of ζ and $\hat{\zeta}' = \zeta'$.

Algorithm ($\mathrm{SWC}^{\sigma,\sigma'}$). The general idea of the algorithm is to guess trisections. Suppose we initially chose blocks B, B' that are the left-most trisections for τ and τ'. Then we obtain an approximation of the prefix of (τ, τ') restricted to B, B' (whichever ends first) by sampling rows of U_B,L_B,$U_{B'}$, and $L_{B'}$. The sampled rows for L_B and $L_{B'}$ provide the interface to the next step. Suppose $L_{B'}$ starts at an earlier row than L_B. Then we guess the trisection of M for τ restricted to the rows of $L_{B'}$ and $X_{B'}$. Let B'' be that block of our algorithm. Then $U_{B''} = L_B$ and we sample rows of $L_{B''}$ in order to approximate a new infix of τ. For a simplified version of the DP without the complications due to having two solution strings, we refer to Appendix A.2.

We globally guess two numbers r and r' that represent $|\tau(M)|$ and $|\tau'(M)|$. We split the processing into an initialization phase and an update phase. In the initialization phase, we assign values to each DP cell (ζ, ζ') based on $\mathrm{SWC}_{\varepsilon^3}$ with the following parameters. We obtain U_i, L_i from the chunks C and U_i', L_i' from the chunks C'. In the execution of $\mathrm{SWC}_{\varepsilon^3}$, we use the selections T, T' instead of trying all possible selections, i.e., T and T' determine all $\tilde{U}_i$, $\tilde{L}_i$, $\tilde{U}_i'$, and $\tilde{L}_i'$ in the algorithm. Let $\tilde{B}$ be the matrix with rows from 1 to the $\min\{c - 1, c' - 1\}$ and columns one to $\min\{\mathrm{end}_B, \mathrm{end}_{B'}\}$. The solution of the

computation is a pair of strings $(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$, the prefixes of the two computed strings until $\text{end}_{\tilde{B}}$. The value of (ζ, ζ') is $\text{cost}_{\tilde{B}}(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$.

In the update phase, we compute the value and the pair of strings of the DP cell (ζ, ζ') as follows. We inductively assume that all DP cells for predecessors of (ζ, ζ') have been updated already. We try all predecessor pairs of DP cells and keep the one that gives the best result (see also Appendix A.2). Let $(\overline{\zeta}, \overline{\zeta}')$ be a predecessor of (ζ, ζ'). By symmetry, we assume without loss of generality that $b' < b$. There are two cases how the two pairs interact. The first case is $\zeta = \overline{\zeta}$. We run $\text{SWC}_{\varepsilon^3}$ on the columns $\text{end}_{\overline{B}'} + 1$ to end_B with the parameters from (ζ, ζ') (see initialization). To obtain the full solution, we append the computed string for B' to the string $\sigma'_{\zeta,\overline{\zeta}'}$ (which is one of the solution strings of the predecessor pair). Let $\tilde{B}$ be the matrix with rows from 1 to the $\min\{c-1, c'-1\}$ and columns one to $\text{end}_{B'}$. The solution of the computation is a pair of strings $(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$, the prefixes of the two computed strings from column one to $\text{end}_{\tilde{B}}$. The potential new value of (ζ, ζ') is $\text{cost}_{\tilde{B}}(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$. We replace the stored solution with the potential new solution if the cost has decreased.

The second case is $\zeta' = \overline{\zeta}$. This case is the crux of the joint DP, since we have a "switch" of the role of σ and σ'.

We run $\text{SWC}_{\varepsilon^3}$ on the columns $\text{end}_{\overline{B}}$ to $\text{end}_{B'}$ with the parameters from (ζ, ζ') (see initialization). To obtain the full solution, we then append the computed string for B to the string $\sigma_{\overline{\zeta},\zeta'}$ (which is one of the solution strings of the predecessor pair). Let $\tilde{B}$ be the matrix with rows from 1 to the $\min\{c-1, c'-1\}$ and columns one to $\text{end}_{B'}$. The solution of the computation is a pair of strings $(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$, the prefixes of the two computed strings until $\text{end}_{\tilde{B}'}$. The potential new value of (ζ, ζ') is $\text{cost}_{\tilde{B}}(\sigma_{\zeta,\zeta'}, \sigma'_{\zeta,\zeta'})$. We replace the stored solution with the potential new solution if the cost has decreased.

For the last strings, we additionally consider special cells that are defined as before, but with $c = n$ or $c' = n$. Intuitively, we use these cells when only at most $1/\varepsilon^4$ rows of $\tau(M)$ or $\tau'(M)$ are left. For pairs of cells containing such ζ or ζ', our computation considers the optimal solution within the computation instead of $\text{SWC}_{\varepsilon^3}$.

Theorem 3.3. The algorithm $\text{SWC}^{\sigma,\sigma'}$ is a PTAS for *SWC*-instances.

Proof. To see that the DP works in polynomial time, we observe that instead of simple DP cells in Lemma A.1 here we consider pairs of DP cells. Therefore the number of cells is squared and thus stays polynomial. During the recursive construction of the solution, we compare each cell to be computed with one compatible cell at a time. Therefore the construction of the solution also takes only polynomial time. As in Lemma A.1, the computed solution is vacuously feasible.

We continue with analyzing the quality of the computed solution. Let (τ, τ') be an optimal solution. We set $r = |\tau(M)|$ and $r' := |\tau'(M)|$. By renaming the two strings we may assume that the last row of the first $(1-\varepsilon^2)r$ rows of $\tau(M)$ is below the first row of the last $\varepsilon^2 r'$ rows of $\tau'(M)$.

We consider DP cells similar to the proof of Lemma A.1. Starting from the top-most row of $\tau(M)$, for each $i \geq 0$, the ith range Y_i contains the next $(\varepsilon^{2i} - \varepsilon^{2i+2})r$ rows of $\tau(M)$. We assign the rows not in $\tau(M)$ such that the first row of each Y_i is contained in $\tau(M)$. Then we choose Y_i such that all rows of M until Y_{i+1} are contained in Y_i.

We consider the DP cells ζ_i for each i with the parameters B_i, C_i, and T_i. The block B_0 contains the rows of Y_0 and Y_1, and the columns one to the end of the first row of $\tau(B_0)$. For each $i > 0$, block B_i contains the rows of Y_i and Y_{i+1}, and the columns after those of B_{i-1} to the end of the first row of B_i.

If only a constant number of rows of $\sigma(M)$ are left, we can compute the partial solutions optimally and there are DP cells for exactly this purpose: there is a DP cell ζ_i such that the last $2/\varepsilon^5$ rows of $\tau(M)$ are located between a_i and c_i and Y_i contains exactly these rows. As before, to keep a clean notation, in the following we implicitly assume that cells with constantly many rows of $\sigma(M)$ are handled separately.

The chunks of C_i are the ranges that equally distribute $\tau(B)$. The selection T_i is the best possible selection as specified in $\text{SWC}_{\varepsilon^3}$. Analogously we define B'_i, C'_i, and T'_i for ζ'_i.

We construct a solution SOL and inductively show that the value of each considered cell (ζ_i, ζ'_j) and (ζ'_i, ζ_j) is at most a factor $(1 + O(\varepsilon))$ larger than the number of errors of an optimal solution restricted to the considered prefix and the considered rows. Afterwards we show that our algorithm computes a solution at least as good as SOL.

We first consider the DP cell (ζ_0, ζ'_0). Recall that we assumed w.l.o.g. that $i'_0 > i_0$. We apply Lemma 3.2 with the parameters of the pair of cells to obtain the prefixes σ_{ζ_0} and $\sigma'_{\zeta'_0}$. The total number of errors within the columns of M at the prefixes is therefore at most a factor $(1 + O(\varepsilon))$ larger than in (τ, τ'). There are two possibilities for the subsequent steps with $i \geq 0$.

We first assume that $b'_{i+1} > b_i$ and consider the cell (ζ_i, ζ'_{i+1}). Then, similar to the proof of Lemma A.1, we apply $\mathrm{SWC}_{\varepsilon^3}$ to obtain the suffix of $(\sigma_{\zeta_i,\zeta'_{i+1}}, \sigma'_{\zeta_i,\zeta'_{i+1}})$ after $\mathrm{end}_{B_{i'}}$. By Lemma 3.2, considering the suffix alone we have at most a factor $(1 + O(\varepsilon))$ more errors within these columns than (τ, τ').

Since ζ_i is a predecessor of ζ_{i+1}, all newly assigned rows were not considered in (ζ_i, ζ'_i). Note that ζ_i did not change. Even though we looked at the same chunks, we used the same selections and therefore did not change $\sigma_{\zeta_i,\zeta'_i}$.

The second possibility is that $b'_{i+1} < b_i$ and we consider the cell (ζ_{i+1}, ζ'_i). The instance is shown in Fig. 3.5. We then apply $\mathrm{SWC}_{\varepsilon^3}$ to obtain the suffix of $(\sigma_{\zeta_{i+1},\zeta'_i}, \sigma'_{\zeta_{i+1},\zeta'_i})$ after end_{B_i}. We obtain a $(1 + O(\varepsilon))$-approximation analogous to the case $b'_{i+1} > b_i$. □

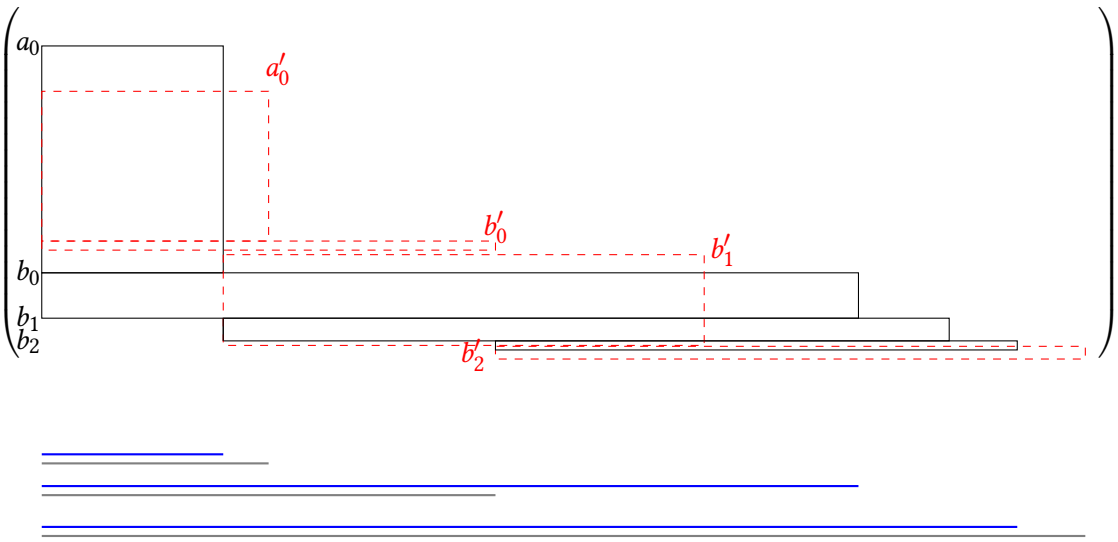

Figure 3.5: Blocks of an instance M in the DP for a pair of solution strings. The blue and gray lines represent σ and σ' respectively from first two iterations of DP. The sketch shows the switch example in the second iteration because $b'_1 < b_0$.

3.6 Subinterval-free instances.

We show how to generalize the results of the previous section in order to handle instances where no interval of a string s is a proper subinterval of a string s' and thus show Theorem 3.2. To this end, we first show how to handle the rooted version of sub-interval free instances, where there is one column j such that each string of the instance crosses j.

We order the rows of a subinterval-free instance M from top to bottom such that for each pair i, i' of rows with the binary part of i starting on the left of the binary part of i', i is above i'. In other words, the binary strings are ordered from top to bottom with increasing starting position (i.e., column). Observe that the sub-string freeness property ensures that the last binary entry of i' is not on the left of the last binary entry of i.

Lemma 3.3. Let M be a Gapless-MEC instance such that no string is the substring of another string. Furthermore we assume that there is a column j of M such that each string of the instance crosses j. Then there is a PTAS for M.

Proof. Let s and t be the first and the last row of M. The column j determines a block W of M that spans all rows and the columns from the first binary entry of t, j_t, to the last binary entry of s, j_s. In particular, W has only binary entries.

The right hand side of j_t (the submatrix of M composed of all columns with index at least j_t) forms a Gapless-MEC instance as required in Theorem 3.3. The submatrix of M that contains all rows of M and columns 1 to j_s forms a Gapless-MEC instance as required in Theorem 3.3 if we invert both the order of the rows and the columns. Instead of changing the ordering of the matrix, we can run the algorithm from right to left and from bottom to top.

We would like to apply Theorem 3.3 independently to the two specified sub-problems. To this end we define a special set of DP cells γ with cells $(\zeta_W, \zeta'_W) \in \gamma$. The content of these cells is similar to the regular cells, but it contains the information for both sides simultaneously. More precisely, a cell ζ_W has the following entrees (see also Figures 3.6 and 3.7).

(a) Three consecutive ranges of rows determined by numbers $1 \leq \overleftarrow{c} < \overleftarrow{b} < \overrightarrow{b} < \overrightarrow{c} \leq n$. These numbers determine an upper range R_U from row $\overleftarrow{b}+1$ to row $\overrightarrow{b}-1$ and the following further ranges. A left lower range $\overleftarrow{R}_L$ from row $\overleftarrow{b}$ to row $\overleftarrow{c}+1$, as well as a right lower range $\overrightarrow{R}_L$ from row $\overrightarrow{b}$ to row $\overrightarrow{c}-1$. (b) A separation into chunks C. There are $3/\varepsilon^2$ chunks in C: $1/\varepsilon^2$ for R_U, $1/\varepsilon^2$ for $\overrightarrow{R}_L$, and $1/\varepsilon^2$ for $\overleftarrow{R}_L$. (c) A selection T of $3/\varepsilon^5$ rows (with repetition): $1/\varepsilon^2$ for each chunk.

We analogously obtain ζ'_W with the same variables but marked with the symbol prime. The rows selected in ζ'_W are required to be disjoint from those in ζ_W, i.e., $T \cap T' = \emptyset$. Also the boundaries of chunks in ζ_W and ζ'_W have to be disjoint.

Definition 3.12 (Center cells). The cells $(\zeta_W, \zeta'_W) \in \gamma$ are called *center cells*.

The reason is that they take a special role as common "centers" of two separate runs of the DP: one run to the left and one run to the right. Observe that for each feasible entry of (ζ_W, ζ'_W), we can apply Theorem 3.3 independently to the left and to the right, since the DP cells (ζ_W, ζ'_W) takes the role of the left-most cell in Theorem 3.3. The strings only overlap between the columns j_t, j_s where we obtain an instance of Binary-MEC, which in particular is a good SWC-instance. Note that for each column $\hat{j}$ on the right hand side of j_t, all rows of W located above $\overleftarrow{b}$ with binary entry at column $\hat{j}$ have also a binary entry at all rows between $\overleftarrow{b}$ and $\overrightarrow{b}$, due to the subinterval-freeness. The properties of $\hat{j}$ on the left hand side of j_s are analogous. We will choose $\overleftarrow{b}$ and $\overrightarrow{b}$ in such a way that by Lemma 3.2, it is therefore sufficient to consider the rows between $\overleftarrow{b}$ and $\overrightarrow{b}$ in order to handle all rows crossing j.

None of the remaining steps from Section 3.5.1 interfere with each other. We therefore run the following *DP*. We first compute all center cells $(\zeta_W, \zeta'_W) \in \gamma$. For each cell, we store an infix of σ and an infix of σ'. The infix of σ starts at j_t and ends at j_s. The entries of the two strings are those that we obtain from $\mathrm{SWC}_{\varepsilon^3}$ with the parameters of (ζ_W, ζ'_W). Each cell (ζ_W, ζ'_W) forms a starting point for Algorithm $\mathrm{SWC}^{\sigma,\sigma'}$, applied independently towards the left hand side and the right hand side.

To see that the DP yields a good enough approximation, again we compare against an optimal solution (τ, τ'). Clearly we get a $(1 + O(\varepsilon))$-approximation for the infix between column j_t and j_s if for a DP cell (ζ_W, ζ'_W), by Lemma 3.2. Note that the computed solution does not consider the rows above $\overleftarrow{c}$ or below $\overrightarrow{c}$. Since the further processing respects our choice between $\overleftarrow{c}$ and $\overrightarrow{c}$, the claim follows from Theorem 3.3. □

General sub-interval-free instances We use Lemma 3.3 to handle general sub-interval free instances. Instead of a single column j crossed by all strings, we determine a sequence $q = (q_1, q_2, \dots)$ of columns with the property that each string crosses exactly one of them. Let s_1 be the first string in M. Then we choose q_1 to be the column of the last entry of s_1.

We recursively specify the remaining columns. For a given j such that we know q_j, let s_i be the last (i.e., bottom-most) string that crosses q_j. Then we choose q_{j+1} to be the last (i.e., rightmost) column of string s_{i+1}. For each q_i in the sequence q, we determine a block W_i analogous to W in Lemma 3.3.

A simple induction shows that by the no-substring property and the chosen order of strings, each string crosses at least one column of q and none of them crosses more than one. In particular, for each j, the solution on the left hand side of q_j depends on rows of M disjoint from the rows that determine the solution on the right hand side of q_{j+1}.

In order to combine the solution on the right hand side of q_j with the solution on the left hand side of q_{j+1}, we introduce a notion of dominance. Let us consider two arbitrary submatrices V_1 and V_2 of M.

Definition 3.13 (Dominance). We say that V_1 τ-dominates V_2 if for each column c that is in both V_1 and V_2, either at least one of the two matrices has no binary entries or the number of binary entries in $\tau(V_1)$ is at least $1/\varepsilon^2$ times the number in $\tau(V_2)$. We say that V_1 is τ-dominant over V_2 for a column c, if the one column submatrix of V_1 determined by c dominates V_2.

We analogously define τ'-dominance.

Consider a submatrix $\overrightarrow{V}$ of M that only contains rows that cross q_i and a submatrix $\overleftarrow{V}$ of M that only contains rows that cross q_{i+1}. We observe that if $\overrightarrow{V}$ is τ-dominant over $\overleftarrow{V}$ for some column c, it is also τ-dominant for all columns on the left hand side of c: until q_i is reached, when moving to the left the number of binary entries of $\tau(\overrightarrow{V})$ increases and the number of binary entries of $\tau(\overleftarrow{V})$ decreases. Analogously, if $\overleftarrow{V}$ is τ-dominant over $\overrightarrow{V}$ for some column c, it is also τ-dominant for all columns on the right hand side of c.

We therefore have a possibly empty interval I without τ-dominance such that the columns of $\overrightarrow{V}$ on the left hand side of I are τ-dominant and the columns of $\overleftarrow{V}$ on the right hand side of I are τ-dominant. (See also Figures 3.6 and 3.7.)

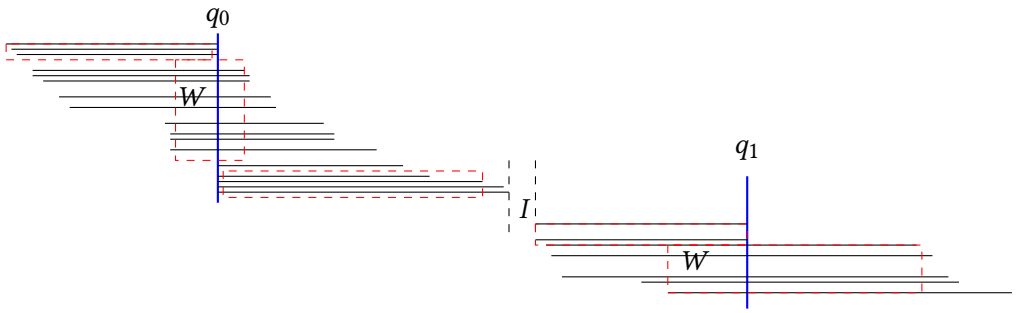

Figure 3.6: Blocks represented by ranges shown in red on an instance M and the blue lines are the columns, I and W shows the empty interval and central region respectively.

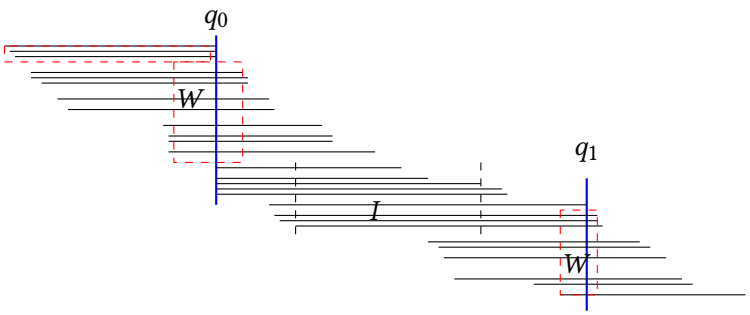

Figure 3.7: This sketch shows a non-dominance example in region I.

Definition 3.14 (Dominance region). The *dominance region* of $\overrightarrow{V}$ with respect to $\overleftarrow{V}$ is the set of columns where $\overrightarrow{V}$ is dominant over $\overleftarrow{V}$, and vice versa.

Within the dominance region, our old DP can simply compute solutions without considering

interferences: the dominated set of rows is small enough to be ignored, applying Lemma 3.2.

Within the interval I, the DP cells on both sides of I have to "cooperate." We obtain a Binary-MEC block in the middle with additional rows on the top and bottom. This sub-instance can be solved directly.

We use DP cells similar to Lemma 3.3, but for more than one center. For each j, we consider column $q_j \in q$ and a collection κ_j of DP cells $(\zeta_{q_j}, \zeta'_{q_j}) \in \kappa_j$. Each cell $(\zeta_{q_j}, \zeta'_{q_j})$ is a center cell with center q_j. We refer to the cells in κ_j as the jth center cells.

Additionally, for each center cell we also store the dominance information on the left and right of q_j, i.e., we store the intervals $\overleftarrow{I}$, $\overleftarrow{I}'$ between q_{j-1} and q_j and the intervals $\overrightarrow{I}$, $\overrightarrow{I}'$ between q_j and q_{j+1} where no cell dominates another, once with respect to τ and once with respect to τ'.

Formally this means to extend the cells by four numbers that store the start and end points four intervals $\overleftarrow{I}$, $\overrightarrow{I}'$, $\overrightarrow{I}$, and $\overrightarrow{I}'$.

For each of the four intervals we store additional information. The four intervals only differ in whether we consider σ or σ'. The left and right version are symmetric. Therefore it is sufficient to analyze the details for a generic $I \in \overleftarrow{I}, \overrightarrow{I}', \overrightarrow{I}, \overrightarrow{I}'$. The interval I determines a block B that we subdivide into chunks C and we select rows T. There are several differences to previous trisections, subdivisions and selections.

We divide the rows of block B into four regions: a middle part $U^{\uparrow}$ that has only binary entries (a Binary-MEC sub-instance) such that each row crosses q_j, a middle part $U^{\downarrow}$ that has only binary entries such that each row crosses q_{j+1}, the rows $U^{\uparrow\uparrow}$ above $U^{\uparrow}$, and the rows $U^{\downarrow\downarrow}$ below $U^{\downarrow}$. We choose the two middle parts such that the number of rows is maximal.

It is not sufficient to use a globally guessed r. Instead, we add four numbers $r^{\uparrow\uparrow}, r^{\uparrow}, r^{\downarrow}, r^{\downarrow\downarrow}$ to the DP cell in order to guess and store the values $\tau(U^{\uparrow})$, $\tau(U^{\downarrow})$, $\tau(U^{\uparrow\uparrow})$, and $\tau(U^{\downarrow\downarrow})$.

Due to the non-dominance, we know that for each column of B, at least an ε^2-fraction of rows from $\tau(B)$ are located in the middle part M. Observe that there is no region that takes the role of X in a trisection. We obtain an instance similar to a good SWC-instance, but it has two non-binary regions and the binary region only has an ε^2 fraction of rows instead of an ε-fraction. To be able to still apply Lemma 3.2, we subdivide each of the three regions into chunks and increase the number of chunks per region. The number of chunks depends on the four versions of r. If our chunks do not contain more than $\varepsilon^4 r$ rows of $\tau(M)$, the lemma is applicable. We guess a number k and set the size of chunks with root q_j to contain $\varepsilon^{4k} r^{\uparrow\uparrow}$ rows of $\tau(M)$ in $U^{\uparrow}$ and $\varepsilon^{4k} r^{\uparrow\uparrow}$ in M for the rows with root q_j. There may be an additional chunk with fewer rows, if the numbers don't match. We specify the remaining chunks symmetrically, based on a number guessed for root q_{j+1}. The choice of k will become clear in the description of the DP.

The increased precision also requires that we increase the precision of the entire remaining DP: we replace each selection of $1/\varepsilon^2$ chunks into selections of $1/\varepsilon^4$ chunks. Clearly, the increased precision cannot decrease the quality of the computed solution.

The idea of the DP is that for each q_j, we run the rooted DP as an *inner* DP that determines solutions for their dominance regions that fit to solutions in the consecutive non-dominance regions. The non-dominance regions then form interfaces that we can use to compute an overall solution from left to right with an *outer* DP.

The inner DP works as follows. For each cell $(\zeta_{q_1}, \zeta'_{q_1}) \in \kappa_1$, we compute the prefixes of σ, σ' until q_1 exactly as in Lemma 3.3. We start the DP to the right hand side also the same way as before, but with the difference that as soon as we reach the row ranges for $\overrightarrow{I}$ or $\overrightarrow{I}'$, we use the choices already stored in $(\zeta_{q_1}, \zeta' q_1)$. We have to ensure that our choices within the DP do not contradict the choices of $(\zeta_{q_1}, \zeta'_{q_1}) \in \kappa_1$. If $(\zeta_{q_1,i}, \zeta'_{q_1,i})$ is the cell of the inner DP that overlaps with $\overrightarrow{I}$ first, we require that among the common rows, $\overrightarrow{B}$ contains the remaining rows from $\tau(M)$ restricted to the rows of the inner DP and does not contradict B_i. Each chunk of C_i contains $\varepsilon^k |\tau(M)|$ rows of $\tau(M)$, for some integer k (the same k that we guessed for the non-domination region). We have to ensure that the chunks of $\overrightarrow{U}^{\uparrow}$ and $\overrightarrow{M}^{\uparrow}$ match the chunks and the chunks of $\overrightarrow{C}_i$. Furthermore, we have to check that the selection of rows matches.

For all $j > 1$ continue in the same manner starting from $(\zeta_{q_j}, \zeta'_{q_j})$ and handle the processing of $\overleftarrow{I}$ as we did before with $\overrightarrow{I}$. Observe that we can see the processing of $(\zeta_{q_1}, \zeta'_{q_1})$ as a special case with empty interval $\overleftarrow{I}$, and to obtain the suffix of σ, σ', the last interval $\overrightarrow{I}$ can be handled as empty interval.

The global DP proceeds from left to right. For each q_j, it considers all cells $(\zeta_{q_j}, \zeta'_{q_j})$. The value of $(\zeta_{q_j}, \zeta'_{q_j})$ is its inner DP value plus the best value achievable on the left hand side with the same choice of parameters for the left non-domination region. Among all cells from κ_j with the same parameters for the right non-domination region, the global DP only keeps the best value (the smallest number of errors).

The above DP is a PTAS for M. Let (τ, τ') be an optimal solution. For each separate $q_j \in q$, we run the same DP as in Lemma 3.3 and thus we obtain a $(1 + O(\varepsilon))$-approximation. For the intervals $\overleftarrow{I}$ and $\overrightarrow{I}$, there is a choice of parameters that matches the choices analyzed in Lemma 3.3. We therefore only have to argue that the transition between sub-instances works correctly. We consider the dominant regions determined by (τ, τ') and consider the DP cells that guess these regions correctly from left to right. Let I be one of the guessed non-dominant regions. We obtain the solution for I by applying $\mathrm{SWC}_{\varepsilon^3}$, which gives a $(1 + O(\varepsilon))$ approximation. The transition between dominant and non-dominant regions uses that in both cases we create the solution strings from the same parameters in $\mathrm{SWC}_{\varepsilon^3}$ and therefore creates the solution from the same instance strings. This finishes the proof of Theorem 3.2.

3.7 A QPTAS for general instances.

To solve the general instances, the main observation is that we divide the rows into their at most $\log_2(m)$ length classes Λ_i, and the ith length class Λ_i is the set of all strings of length ℓ with $\ell \in (m/2^{i+1}, m/2^i]$. First we present an algorithm to solve each length class Λ_i separately by constructing their corresponding columns.

3.7.1 Length classes.

We show how we can handle length classes of strings. To this end, let us assume w.l.o.g. that m (i.e., the number of columns in M) is a power of 2. Then for each $i \geq 0$, the ith length class Λ_i is the set of all strings of length ℓ with $\ell \in (m/2^{i+1}, m/2^i]$. We observe the following known property of length classes.

Lemma 3.4. For each $i \geq 0$ there is a set $q_i = \{q_{i,1}, q_{i,2}, \dots\}$ of columns such that (a) each string in Λ_i crosses at least one column from q_i and (b) no string from Λ_i crosses more than two columns from q_i. Furthermore, we can choose the sets such that $q_i \subseteq q_{i+1}$.

Proof. At level i, for each k with $1 \leq k \leq 2^{i+1}$ we select the column with index $k \cdot m/2^{i+1}$. We observe that the distance between two consecutive columns from q_i is $m/2^{i+1}$, which matches the shortest length of strings in Λ_i: if a minimal string starts right after a column of q_i, its last entry will cross the next column of q_i.

Since strings do not start before column 1 and column m is contained in each q_i, claim (a) follows. To see (b), observe that a maximum length string of Λ_i is at most $m/2^i$. Let j be an index. The number of columns from $q_{i,j}$ to the column right before $q_{i,j+1}$ and from $q_{i,j+1}$ to right before q_{i+2} are exactly $m/2^{i+1}$. If the string starts directly at a column $q_{i,j}$ from q_i, it would cross column $q_{i,j+1}$ and end right before column $q_{i,j+2}$.

The last claimed property follows directly from the construction of the sets q_i. (See also Fig. 3.2). □

For each i, we now separate Λ_i into two sub-instances. One sub-instance Λ'_i is formed by those rows from Λ_i that only cross one column of q_i and the second sub-instance Λ''_i is formed by those rows that cross exactly two columns of Λ_i.

Definition 3.15 (DP for a length class Λ_i). For each index j let ξ'_j be the sets of DP cells for Λ'_i and for the odd indices j let ξ''_j be the set of cells for Λ''_i. We define a super-cell that starts in j, $(Z'_j, Z''_j, Z'_{j+1}, Z''_{j+2}) \in \xi'_j \times \xi''_j \times \xi'_{j+1} \times \xi''_{j+2}$ and the super-cell that ends in j, $(Z'_{j-1}, Z''_{j-2}, Z'_j, Z''_j) \in \xi'_{j-1} \times \xi''_{j-2} \times \xi'_j \times \xi''_j$.

Lemma 3.5. There is a QPTAS for Gapless-MEC if all strings are in the same class Λ_i.

To prove it, we consider DP-cells according to Definition 3.15 and combine these cells from two consecutive columns such that they are compatible.

Proof. To combine the PTAS for Λ'_i and Λ''_i, we proceed from left to right. For each index j let ξ'_j be the sets of DP cells for Λ'_i and for the odd indices j let ξ''_j be the set of cells for Λ''_i. For each column c before $q_{i,1}$, each pair of cells $(Z, Z') \in \xi'_1 \times \xi''_1$ determines two subproblems for which we compute the two separate solutions. Let $\mathcal{B}(Z, Z', z)$ be the set of all boxes (sets of rows) for σ considered in the sub-cells of (Z, Z') at column z and let $\mathcal{B}'(Z, Z', z)$ be the set of all boxes (sets of rows) for σ' considered in the sub-cells of (Z, Z') at column z.

We now extend the DP as follows. We compose the solution from left to right, starting with the prefix of (σ, σ') before $q_{i,1}$ and then, step by step, we fill the intervals between $q_{i,j}$ and $q_{i,j+1}$ for $j \geq 1$. The starting interval can be seen as the interval between a dummy-column q_0 and q_1. For each j, let us analyze its interval. If j is odd, we simultaneously consider the cells $\xi'_j, \xi'_{j+1}, \xi''_j, \xi''_{j+2}$. Otherwise, we simultaneously consider the cells $\xi'_j, \xi'_{j+1}, \xi''_{j-1}, \xi''_{j+1}$.

For each column c with index ℓ in the interval, in both cases the values of the DP cells reveal all $\mathrm{SWC}_{\varepsilon^3}$ instances at c that we would have to solve in order to obtain solutions for Λ'_i and Λ''_i separately. Instead of solving these instances separately, we solve them simultaneously.

Let $\hat{C}_1, \hat{C}'_1$ and $\hat{C}_2, \hat{C}'_2$ be the chunks of the four DP cells at position j. In order to determine the value σ_j, we have to combine $\hat{C}_1$ with $\hat{C}_2$ and take care of the different densities of rows. To this end, we generalize the function $\textsc{Majority}_j$.

Definition 3.16. Generalized Majority For a single chunk c, let $r(c)$ be the number of rows in c that are guessed to be in $\tau(M)$ and $t(c)$ the number of selected rows. As in Definition 3.5, we replace all values zero by -1. Then, for the given set of chunks C with selection T, we compute

$$\rho := \sum_{c \in C} \sum_{i \in T \colon i \in c} (r(c) \cdot M_{i,j}) .$$

We set $\sigma_j = 1$ if the outcome is at least zero and 0 otherwise. The definition is analogous for σ'_j.

By replacing the majority function by the generalized majority function of Definition 3.16 in the proof of Lemma 3.1, we obtain a $(1 + O(\varepsilon))$-approximation also if we consider different cells simultaneously.

Since we consider all cells for the entire interval simultaneously, one of the choices is at least as good as sampling uniformly at random with knowledge of $\tau(M)$ and $\tau'(M)$. We therefore obtain a solution for the interval with at most a $(1 + O(\varepsilon))$ factor of errors compared to (τ, τ')

Finally we have to join the results that we obtain for the intervals. Observe that for each pair of cells $(Z''_j, Z''_{j+2}) \in \xi''_j \times \xi''_{j+2}$ there are two consecutive pairs of cells $(Z'_j, Z'_{j+1}) \in \xi'_j \times \xi'_{j+1}$ and $(Z'_{j+1}, Z'_{j+2}) \in \xi'_{j+1} \times \xi'_{j+2}$. For a quadruple of cells $(Z'_j, Z''_j, Z'_{j+1}, Z''_{j+2})$ we consider each quadruple on the left hand side ending with the matching cells Z'_j, Z''_j. Among these, we take the one with fewest errors. To obtain the value of the new quadruple, we add the errors in the interval $(q_{i,j} q_{i,j+1}]$ to the value of the selected predecessor quadruple. To compute the value $(Z'_{j+1}, Z''_j, Z'_{j+2}, Z''_{j+2})$, we consider all cells $(Z'_j, Z''_j, Z'_{j+1}, Z''_{j+2})$, i.e., the cells that have the same $Z''_j, Z''_{j+1}, Z'_{j+1}$ for all choices of Z'_j. We add the errors between $q_{i,j+1}$ and $q_{i,j+2}$ to the smallest value found among the predecessors.

The approximation ratio follows from Lemma 3.6 and the quasi-polynomial running time from the fact that we only consider constantly many super-cells simultaneously.

□

Lemma 3.6. There is a QPTAS for Gapless-MEC if all strings are in the same class Λ'_i or Λ''_i.

Proof. We first note that by skipping all $q_{i,j}$ with even j, the strings in Λ_i'' cross exactly one column of the set. It is therefore sufficient to handle Λ_i'.

For each column $q_{i,j}$, we create a set of DP cells (as defined in Definition 3.10) that stores information about a center region as defined in Definition 3.12 and about non-domination intervals as defined in Definition 3.13, exactly as in the proof of Theorem 3.2. The next insight is that we can order the rows crossing column c at the left and right side as defined below.

Left row ordering We first order the rows with increasing starting positions of strings as in Lemma 3.3. At the left side of column c , we obtain a similar instance as in Lemma 3.3.

Right row ordering Afterwards we reorder the rows in order to handle the right hand side of column c. More precisely, we order the strings in increasing order based on the end of strings e_i. The obtained structure corresponds to the right hand side of column c is similar to the instance handled in Lemma 3.3.

Instead of running the DP of Lemma 3.3, we guess the sequence of blocks. An optimal solution (τ, τ') determines a sequence of blocks $\overleftarrow{A}_1, \overleftarrow{A}_2, \ldots, \overleftarrow{A}_k$ such that $|\tau(\overleftarrow{A}_{i+1})| = \varepsilon^2|\tau(\overleftarrow{A}_i)|$. Instead of moving from $\overleftarrow{A}_1$ to $\overleftarrow{A}_k$ using a DP, we directly guess the strings for all k sub-matrices *simultaneously*. We do the same with the chunks and row selections. Additionally, we guess the sequence of sub-matrices $\overleftarrow{A}'_1, \overleftarrow{A}'_2, \ldots, \overleftarrow{A}'_{k'}$ simultaneously such that $|\tau'(\overleftarrow{A}'_{i+1})| = \varepsilon^2|\tau'(\overleftarrow{A}'_i)|$. We obtain a combined DP cell $\overleftarrow{\zeta}$ for $k + k'$ sub-matrices.

Again we form the sub-matrices $\overrightarrow{A}_1, \overrightarrow{A}_2, \ldots, \overrightarrow{A}_k$ and $\overrightarrow{A}'_1, \overrightarrow{A}'_2, \ldots, \overrightarrow{A}'_{k'}$ analogous to the left hand side and guess the selected strings of all matrices simultaneously such that $|\tau(\overrightarrow{A}_{i+1})| = \varepsilon^2|\tau(\overrightarrow{A}_i)|$ and $|\tau'(\overrightarrow{A}'_{i+1})| = \varepsilon^2|\tau'(\overrightarrow{A}'_i)|$. We obtain a combined DP cell $\overrightarrow{\zeta}$ for all $k + k'$ sub-matrices on the right hand side.

Definition 3.17 (DP cell for sub-class of a length class Λ_i). For each $q_{i,j}$, let ξ_j be the set of super-cells $(\overleftarrow{\zeta}, \overleftarrow{\zeta}', \overrightarrow{\zeta}, \overrightarrow{\zeta}')$, but with the additional center and non-domination information of Lemma 3.3.

For each $q_{i,j}$, let ξ_j be the set of super-cells based on Definition 3.17. We then design a DP that moves from left to right through the columns in q_i. The DP and its analysis now follow from the proof of Theorem 3.2, but we consider the left hand side and right hand side of each cell from ξ_j simultaneously.

To analyze the running time, we observe that k and k' are at most $O(\log_{1/\varepsilon}(n))$ since for each i we assume that $|\tau(\overleftarrow{A}_{i+1})| = \varepsilon|\tau(\overleftarrow{A}_i)|$ and $|\tau'(\overleftarrow{A}'_{i+1})| = \varepsilon|\tau'(\overleftarrow{A}'_i)|$. The number of instances $\overrightarrow{A}_i$ and $\overrightarrow{A}'_i$ are also at most $O(\log_{1/\varepsilon}(n))$ each, for the same reason.

We thus obtain super-cells that are combined of logarithmically many sub-cells with polynomial complexity. We obtain an overall super-cell which is a quadruple $(\overleftarrow{\zeta}, \overleftarrow{\zeta}', \overrightarrow{\zeta}, \overrightarrow{\zeta}')$, and we have to distinguish $\left(n^{O(1)}\right)^{4\log_{1/\varepsilon}(n)} = n^{O(\log n)}$ different cells, which is quasi-polynomial[2].

We now analyze the performance guarantee. For each column j, we obtain the values σ_j and σ'_j in almost the same way as we do in Lemma 3.3, but with the difference that we require consistency with all other rows sampled. For an optimal solution (τ, τ'), it is sufficient to only consider choices of rows such that all rows selected for σ are in $\tau(M)$ and all rows selected for σ' are in $\tau'(M)$. Such a selection of rows ensures consistency. Note that we could apply the proof of Lemma 3.3 from the root to the left hand side and to the right hand side independently, if we knew $\tau(M)$ and $\tau'(M)$, just by avoiding wrong assignments. The simultaneous selection of all relevant rows ensures that we consider at least one selection of rows that satisfies these strong conditions. This solution is a $(1 + \varepsilon)$ approximation by the proof of Lemma 3.3, and our DP computes a solution of at least the same quality since we consider the overall number of errors with respect to all sampled rows. □

Combining the two sub-classes gives a QPTAS for an entire length class.

[2]We assume that n and m are polynomially related. This is justified because there are $n \cdot m$ entries of M and therefore measuring in m instead of n would also give a quasi-polynomial complexity.

3.7.2 The general QPTAS.

Finally we combine our insights to an algorithm for general instances by combining different length classes. (See also Fig. 3.1.)

For different length classes Λ_i, we construct their corresponding columns as explained in the previous section. The main idea is that for each column j, we only have to consider those quadruple of super-cells according to Definition 3.15 that cross j from all the length classes simultaneously. We therefore consider at most $O(\log(n))$ quadruples of super-cells simultaneously. In the dynamic program, we consider a joint quadruple of super-cells from all the length classes. Then the overall complexity of a joint cell is quasi-polynomial: the number of different cells is $\left(n^{O(\log n)}\right)^{O(\log n)} = n^{O(\log^2 n)}$.

Let $Q_{i,j}$ be the set of quadruples of length class i crossing column j such that the strings are ordered from shortest length class to the longest. For each length class i, a quadruple $q \in Q_{i,j}$ is the set of rows starting at j, cross j, or end in j. If j is the index of $q_{i,\ell}$, the quadruple q starts in j if it is formed by cells $(Z'_\ell, Z''_\ell, Z'_{\ell+1}, Z''_{\ell+2})$ and ends in j if it is formed by $(Z'_{\ell-1}, Z''_{\ell-2}, Z'_\ell, Z''_\ell)$ (see Definition 3.15). If j lies between $q_{i,\ell}$ and $q_{i,\ell+1}$, j crosses those quadruples that contain Z'_ℓ and $Z'_{\ell+1}$. If non of the cases are true, we do not consider q in the cells for column j.

Let us consider a $\log(n)$ vector of quadruples v, with one quadruple $Q_{i,j}$ for each i and, consider quadruples starting at, ending at, or crossing column j for length class i. We require that if for some i, the quadruple $q \in Q_{i,j}$ ends at j, then for all the length classes Λ_k with $k > i$ the same condition holds (with index larger than i). This also implies that if for some i, the quadruple of length class i starts at j, then the same also holds for all quadruples of shorter length classes (with index larger than i). In particular, in order to be able to combine neighboring vectors of quadruples, we do not allow to mix starting and ending quadruples. Let ϕ be the set of all $\log(n)$ vectors of tuples as described above (with one tuple of each length class). The tuple for each length class is defined as in Lemma 3.5 and the DP for general instances follows the ideas of Lemma 3.5: We move from left to right column by column. In the initialization step, the joint DP cell is initialized based on Algorithm 1 using ϕ. We guess the blocks, chunks and selections from each length class and consider them jointly in a DP cell.

For column j, let us consider a vector $v \in \phi$. We distinguish whether v has starting or ending quadruples. (One of the two cases must apply due to the shortest length class.) For a $v \in \phi$ with starting quadruples, let d be the smallest number such that there is a quadruple of length class d starting at j. To compute v we consider all $v' \in \phi$ with the following properties. (a) v' has the same quadruples for all length classes $d' < d$ and (b) for $d' \geq d$, the right hand sides of the quadruples of length class d' in v' compatible the left hand sides of the quadruples of v. The super-cells from the left and right hand side are compatible if the intersecting strings from the left and right hand side are assigned to the same types of solution string σ or σ'.

For a $v \in \phi$ with ending quadruples, let d be the smallest number such that there is a quadruple of length class d ending at $j-1$. (In the very first column of the instance, we do not need this value.) To compute v we consider all $v' \in \phi$ with the following properties. (a) v' has the same quadruples for all length classes $d' < d$ and (b) for $d' \geq d$, the right hand sides of the quadruples of length class d' in v' match the left hand sides of the quadruples of v in column $j-1$. Then the value of v is the sum of the minimum value over all such v' and the number of errors in column j obtained by applying $\mathrm{SWC}_{\varepsilon^3}$ exactly as in the proof of Lemma 3.5.

The approximation ratio follows by arguing that the expected number of errors at each column is at most $(1 + O(\varepsilon))$ of OPT (see Lemma 3.5). This finishes the proof of Theorem 3.1.

3.8 Discussion

The approximation status of Gapless-MEC instances has been open for 10 years (Cilibrasi et al., 2007). We have studied the problem by unraveling its structures and identifying cases where the PTAS algorithm for Binary-MEC can not be directly applied for Gapless-MEC. Based on the ideas of the algorithm for Binary-MEC, we build our own dynamic programming algorithm, by additionally including some deep insights. We have proved that the dynamic programming algorithm is in QPTAS. Proving that the Gapless-MEC in QPTAS rules out the possibility of this problem to be in APX-hard and also provides a hint that this problem can exhibit a PTAS.

This chapter has a preprint (Garg and Mömke, 2018), which has a co-author as T. Mömke. The figures in this chapter are taken from this preprint. I co-invented this approximation algorithm with T. Mömke and co-wrote the paper with him.

Chapter **4**

Parameterized algorithm for phasing individual genomes

In the previous chapter, we studied the approximation status of MEC and its variants. In this chapter, we first explore the practical approaches to solve the MEC and present the literature survey on these approaches. Additionally, we survey practical approaches for solving other problem formulations of haplotype assembly such as MLF and MFC introduced in Chapter 1. Afterwards, we highlight the importance of combining multiple sequencing technologies to generate complete haplotypes for individual genomes. We further explore how to use the MEC formulation for this data integration.

4.1 Literature survey

Table 4.1 presents the summary of practical approaches for solving MEC and other problem formulations. Wang et al. (2005) provides the exact algorithm for solving the MEC model. They model a haplotype problem as a binary tree and haplotypes are viewed as the optimal path in this tree. They apply a branch and bound algorithm to solve this haplotype problem. This algorithm searches an optimal path in a binary tree, in which the node on the j-th level denotes the j-th fragment and the branch on the path connecting its child denotes its corresponding haplotype assignment. A binary string is used to represent an assignment of each fragment to one of the two haplotypes (a feasible solution to the MEC model). The algorithm starts from root node by adding the first fragment and calculates the MEC score. Then, if the calculated score is bigger than the previous score, the node will be divided. This process is continued until all the fragments are considered. This results in a binary tree and then the optimal haplotype path is computed in this tree. The branch and bound algorithm can find the exact optimal solution, but the running time is exponential in the number of fragments. Therefore, it is not useful on large datasets.

Lim et al. (2012) first identify an initial upper bound using a local search algorithm and reduce the search space of the branch and bound algorithm based on this computed upper bound. By using recursive property of bounding functions with some lookup table, they can solve the MEC problem in reasonable time. Wang et al. (2012) consider the original fragments and uses Hamming distance to calculate the difference between a haplotype and a fragment. They use the genetic algorithm and their approach works even for tri- or tetra-allelic loci and homozygous sites.

Another approach for solving haplotyping using the MEC is to reduce it to a satisfiability (SAT) problem. He et al. (2010) propose a partial Max-SAT formulation for haplotype assembly. Solvers such as Clone and WBO are used to solve the resulting Max-SAT problems. Mousavi et al. (2011) suggest a Max-2-SAT problem, which is more general than the partial Max-SAT. The general Max-SAT solver is used for solving the instances. Thus their formulation is more generalized and their approach works even for homozygous alleles that can appear due to sequencing errors. Also, their Max-SAT formulation

Formulation	Approach	Authors
MEC	Branch and Bound	Wang et al. (2005), Lim et al. (2012)
	Genetic algorithm	Wang et al. (2005), Wang et al. (2012)
	Satisfiability (SAT)	Mousavi et al. (2011), He et al. (2010)
	Probabilistic approach	Chen et al. (2008), Bansal et al. (2008), Bansal and Bafna (2008), Kuleshov (2014)
	Parameterized	Deng et al. (2013), Pirola et al. (2015), Patterson et al. (2015)
	ILP	Chen et al. (2013)
	Clustering	Wang et al. (2007)
MLF	Dynamic Programming	Zhao et al. (2005), Xie et al. (2008), Kang et al. (2010), Wu et al. (2013)
MFC	Graph	Duitama et al. (2010)
Others	Mixture Model	Matsumoto and Kiryu (2013)
	Heuristic dynamic programming	Xie et al. (2012)
	Graph (spanning tree)	Aguiar and Istrail (2012), Mazrouee and Wang (2014)

Table 4.1: Related work on computational approaches to haplotyping for a single individual

results in instances with fewer variables and clauses than those of the Partial Max-SAT formulation. The heuristic Max-SAT solver irots provided in the UBCSAT package is used.

Some studies tried to model haplotyping problems by using probabilistic approaches. Since the fragments of the input SNP matrix stem from two haplotypes, Chen et al. (2008) assumed that the fragments were generated according to two parameters, representing sequencing errors and incompleteness errors. They also assumed a third parameter denoting a minimum difference between two haplotypes. When the parameters were unknown, they assumed the existence of those parameters from the input SNP matrix and can reconstruct the haplotypes with high probability. Thus they designed a probabilistic function using those parameters for the haplotype. As a result, the fragments from the fragment matrix $\mathcal{F}$ were divided into two sets and the most frequent character in each SNP site was selected to determine the haplotype sequence. HASH by Bansal et al. (2008) and HAPCUT by Bansal and Bafna (2008) also used probabilistic models based on graph structure. They constructed a graph from the input matrix, where the nodes correspond to the columns of the matrix. If there is a fragment that includes two sites, the two nodes are connected by an edge. The weight of the edge is the number of fragments inconsistent with the current phase between the SNP pair minus the number of fragments consistent with the phase. Informally, the weight represents the weakness of phasing in SNP pair. HASH used a graph-cut algorithm and constructs a Markov chain, but HAPCUT optimized the MEC score by using a Max-Cut algorithm. Using a greedy algorithm, the final pair of haplotypes was determined based on the resulting MEC score. Although HASH and HapCut achieve reasonably good results, they are heuristics and therefore can not guarantee optimal solutions.

Another method used dynamic programming to determine the haplotypes for a single individual (He et al., 2010). For the input reads encoded as the usual fragment matrix, this approach first solves partial instances optimally and then extends the partial haplotypes by one bit repeatedly to obtain full-length haplotypes. They showed that method can be applied to whole-genome sequencing datasets. However, the method does not scale with increasing SNP sites per read because the running time is $O(m \cdot 2^k \cdot n)$, where m is the number of reads, k is the length of longest read and n is the total number of SNPs in the haplotypes. To solve this drawback, Deng et al. (2013) designed a different dynamic programming method whose running time is $O(n \cdot 2^t \cdot t)$, where t is the maximum coverage and n is the number of columns. For large MEC instances, they combined this dynamic programming with a heuristic approach. This method first applies a heuristic to obtain a subset of the input matrix $\mathcal{F}$

by using a randomized sampling approach and then carries out the dynamic programming. Once it produces an initial solution from the submatrix, it refines the haplotypes by comparing the solution to all fragments. By repeatedly solving a submatrix and refining the results, the final haplotypes are determined. More recently, Patterson et al. (2015) provided an FPT algorithm to solving the wMEC model. The running time is $O(2^c ov \cdot m)$, where cov is the maximum coverage and m is the number of columns. Pirola et al. (2015) considered a restricted variant of MEC, in which up to k corrections were allowed per SNP position, and presented an FPT algorithm that runs in time exponential in k. The running time is $O(cov^{k+1} \cdot L \cdot m)$, where cov is the maximum coverage, L is the read length and m is the number of SNP positions.

In another approach, the first ILP formulation for MEC was given by Fouilhoux and Mahjoub (2012) and was based on a reduction to the maximum bipartite induced sub-graph problem. In another ILP approach by Chen et al. (2013), the binary variable x_k for column $\mathcal{F}(k)$ was considered such that its value is supposed to be 1, if and only if the k^{th} bits of h^0 and h^1 are 1 and 0 respectively. Moreover, the binary variable y_j for row $\mathcal{F}(j)$ was considered such that its value is supposed to be 1, if and only if the read corresponding to $\mathcal{F}(j)$ is aligned to h^0 and otherwise 0. The objective function was to minimize the number of flips in each entry from all the rows such that all rows can be assigned to the original haplotypes h^0 or h^1 without any conflict.

Another approach by Wang et al. (2007) considered a clustering algorithm that is used to split the rows of $\mathcal{F}$ in two sets. The main contribution consists in the combination of the two distance functions used by the clustering algorithm. The first distance is the Hamming distance as defined in Equation 1.1, which basically computes the number of mismatches between two fragments. The second distance D' considers the number of matches between the two fragments. The main idea is based on the intuition that, given a certain fixed number of mismatches between two fragments, the more they overlap the closer they are. Using the above distance functions, a simple iterative clustering procedure is given as follows. The hamming distance is computed for each possible pair of fragments in the SNP matrix. Let the two fragments r_1 and r_2 have the highest Hamming distance and the clusters are initialized as $C1 = r_1$ and $C2 = r_2$. Let the computed consensus strings are H1 and H2 from these two clusters C1 and C2 such that all the fragments are compared with H1 and H2 and assigned to the corresponding closer set. The ambiguity in the assignment of fragments to both consensus strings is broken based on the distance D'. Once all fragments are assigned, the consensus strings H1 and H2 are updated and the algorithm iterates. The procedure loops until a stable haplotype pair is found (i.e. when the consensus haplotypes are the same before and after the update).

There were some approaches to solve other objective functions such as MFR and MSR. They mainly used graphs, dynamic programming and other heuristic methods to solve haplotyping problem.

Thus a number of computational approaches for read-based phasing have been developed and, particularly, progress on fixed-parameter tractable (FPT) algorithms has enabled solving read-based phasing in practice, for instance through the implementations available in the software package WhatsHap[1] (Martin et al., 2016), distributed as Open Source software under the terms of the MIT license.

4.2 WhatsHap Algorithm

WhatsHap (Patterson et al., 2015) is a Dynamic programming (DP) algorithm to optimally solve the wMEC problem (Problem 1.2. It runs in $\mathcal{O}(2^c \cdot M)$ time, where M is the number of variants to be phased and c is the maximum physical coverage (which includes internal segments of paired-end reads). Since it is independent of the read-length, so it is suitable for sequencing technologies producing long reads. The general idea is to proceed column-wise from left to right while maintaining a set of active reads. Each read remains active from its first non-dash position to its last non-dash position in $\mathcal{F}$. Let the set of active reads in column k be denoted by $A(k)$. Note that $c = \max_k\{|A(k)|\}$. For each column k of $\mathcal{F}$, we fill a DP table column $C(k, \cdot)$ with $2^{|A(k)|}$ entries, one entry for each bipartition B of the set of active

[1] https://bitbucket.org/whatshap/whatshap

reads $A(k)$. Each entry $C(k, B)$ is equal to the cost of solving wMEC on the partial matrix consisting of columns 1 to k of $\mathcal{F}$ under the assumption that the sought bipartition of the full read set $A(1) \cup \ldots \cup A(k)$ *extends* B according to the below definition.

Definition 4.1 (Bipartition extension). For a given set A and a subset $A' \subset A$, a bipartition $B = (P, Q)$ of A is said to *extend* ($\simeq$) a bipartition $B' = (P', Q')$ of A' if $P' \subset P$ and $Q' \subset Q$.

By this semantics of DP table entries $C(k, B)$, the minimum of the last column $\min_B\{C(M, B)\}$ is the optimal wMEC cost.

DP cell initialization. Along similar lines as Patterson et al. (2015), we first compute the local cost incurred by bipartition $B = (R, S)$ in column k, denoted $\Delta_C(k, B)$, and later combine it with the corresponding costs incurred in previous columns. The cost $W^a_{k,R}$ of flipping all entries in a read set R to the same allele $a \in \{0, 1\}$ is given by

$$W^a_{k,R} = \sum_{j \in R} [\![\mathcal{F}(j, k) \neq a]\!] \cdot \mathcal{W}(j, k),$$

In the same manner, we can compute costs $W^a_{k,S}$ for read set S to some allele a.

So given the corresponding column vectors $\mathcal{F}(k)$ and $\mathcal{W}(k)$ of the MEC matrix and of the weight matrix, respectively, and the bipartition $B = (R, S)$ of active reads $A(k)$, the cost $\Delta_C(k, B)$ is computed by:

$$\Delta_C(k, B) = \min \{ W^0_{k,S} + W^1_{k,R}, W^1_{k,S} + W^0_{k,R} \}, \tag{4.1}$$

where minimization considers the two possibilities of assigning those two alleles to the two haplotypes.

DP column recurrence. Note that $C(k, B)$ is the cost of an optimal solution of Problem 1.2 for input matrices restricted to the first k columns under the additional constraint that the solution's bipartition of the full read set extends B. Since column k lists all bipartitions, the optimal solution to the input matrix consisting of the first k columns would be given by the minimum in that column. To compute entries in column $C(k + 1, \cdot)$, we add up local costs incurred in column $k + 1$ and costs from the previous column is given as:

$$C(k + 1, B) = \Delta_C(k + 1, B) + \min_{B' \in \mathcal{B}(A(k)) : B \simeq B'} C(k, B') \tag{4.2}$$

where $\mathcal{B}(A(k))$ denotes the set of all bipartitions of $A(k)$.

To adhere to the semantics of $C(k+1, B)$ described above, only entries in column k whose bipartitions are *compatible* with B are to be considered as possible "predecessors" of $C(k + 1, B)$. Finally, we can backtrace from the last column to get the haplotypes.

Let us consider an example to understand how the algorithm works. We consider an example SNP matrix $\mathcal{F}$ and its corresponding weight matrix $\mathcal{W}$ as follows.

$$\mathcal{F} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{cc} v_1 & v_2 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 1 \end{pmatrix} \end{array} \tag{4.3}$$

$$\mathcal{W} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{cc} v_1 & v_2 \end{array} \\ \begin{pmatrix} q_1 & q_2 \\ q_3 & q_4 \\ q_5 & q_6 \end{pmatrix} \end{array} \tag{4.4}$$

Let us consider some values in the weight matrix (4.4):

$$\mathcal{W} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{cc} v_1 & v_2 \end{array} \\ \begin{pmatrix} 3 & 10 \\ 9 & 1 \\ 4 & 5 \end{pmatrix} \end{array} \tag{4.5}$$

So the goal is to partition the reads into two sets with minimum flipping cost. If we try in a brute-force manner, the possible bipartitions are as follows.

- For bipartition $(\{r_1\}, \{r_2, r_3\})$, the cost is 1. This is achieved by flipping entry $\mathcal{F}(2,2)$ with a cost $\mathcal{W}(2,2) = 1$.
- For the bipartition $(\{r_1, r_2, r_3\}, \emptyset)$, the cost computes to 14. This is achieved by flipping entries $\mathcal{F}(1,1), \mathcal{F}(1,2), \mathcal{F}(2,2)$, with a cost of 3+10+1 = 14.

In a similar manner, we can compute the cost for other possible bipartitions.

Since this example is very small, it is doable in a brute-force manner. For large input instances, we explain the FPT algorithm implemented in WhatsHap. As explained above, the initialization for first column can be computed as follows. We consider all possible bipartitions and store cost for each bipartition.

For instance, the first DP column for the above example for different bipartitions, $\Delta_C(1,.)$ can be filled in a following way using Equation (4.1):

$$C(1, (\{r_1, r_2, r_3\}, \emptyset)) = \min\{3 + 0, 13 + 0\} = 3$$

Similarly, we can compute $\Delta_C(1,.)$ for other bipartitions $(\{r_1, r_2\}, \{r_3\}), (\{r_1, r_3\}, \{r_2\})$, $(\emptyset, \{r_1, r_2, r_3\}), (\{r_3\}, \{r_1, r_2\}), (\{r_2\}, \{r_1, r_3\})$.

Now, let the DP table entries in the second column $C(2,.)$ for different bipartitions by using Equation (4.2):

$$C(2, (\{r_1, r_2, r_3\}, \emptyset)) = \min\{11 + 0, 5 + 0\} + \min\{C(1, (\{r_1, r_2, r_3\}, \emptyset)\} = 5 + 3 = 8$$

To fill DP column $C(2,.)$, we can analogously compute the cost for the remaining bipartitions $(\{r_1, r_2\}, \{r_3\})$, $(\{r_1, r_3\}, \{r_2\})$, $(\emptyset, \{r_1, r_2, r_3\})$, $(\{r_3\}, \{r_1, r_2\})$, and $(\{r_2\}, \{r_1, r_3\})$.

Once we reach the last column and know the optimal bipartition, we can backtrace to get the corresponding haplotypes. We observe that the running time is linear in the number of variants and independent of read length.

4.3 The need for combining different sequencing technologies

All approaches to reconstruct haplotypes from sequencing reads, reference-based or reference-free, come with the intrinsic limitation that the distance between subsequent heterozygous markers can be larger than the read length itself. While long-read sequencing (such as PacBio SMRT (Steinberg et al., 2014) and Oxford NanoPore MinION (Ammar et al., 2015)), or linked read data (such as those provided by 10X Genomics (Zheng et al., 2016)) help to mitigate this issue, these technologies fail to phase over longer stretches of homozygosity, repeat-rich areas including segmental duplications, and centromeres. Thus, specialized techniques that enable homologous chromosomes to be discriminated are required to physically connect alleles across whole chromosomes (Zheng et al., 2016; Ma et al., 2010; Yang et al., 2011). As an alternative to whole chromosome separation, chromatin capture (Hi-C) methods (Lieberman-Aiden et al., 2009) can be employed to infer long-range haplotype information, based on the assumption that a chromosome will be cross-linked to itself more often than to its homologue (Selvaraj et al., 2013). Recently, Hi-C data sets have been used in combination with other sequencing methods for long-range phasing (Edge et al., 2017; Ben-Elazar et al., 2016). However, it has been shown that to generate a reliable long-range haplotype scaffold, relatively high sequence coverage (ideally 90-fold) is needed to reduce bias caused by cross-links between non-homologous chromosomes (Edge et al., 2017). In particular, because these haplotypes need to be inferred statistically, the probability that two heterozygous variants are correctly phased relative to each other, deteriorates with increasing chromosomal distances.

Our aim is to obtain dense and global haplotypes that span centromeres, homozygosity regions and genome assembly gaps, while keeping error rates, costs and labor at minimum. To this end, we harness the long-range phasing information provided by single cell template strand sequencing (Strand-seq) (Falconer et al., 2012; Sanders et al., 2017). Strand-seq is an effective method to assemble highly accurate chromosome-length haplotypes, albeit with lower density of phased alleles in comparison to read-based

phasing (Porubský et al., 2016). Unlike other haplotyping methods, Strand-seq, by design, distinguishes parental homologues based on the directionality of single-stranded DNA. Therefore, Strand-seq is able to deliver global haplotypes, and its capability to correctly phase two variants with respect to each other does not depend on their distance. To fully exploit this advantage, while at the same time generating dense haplotypes that contain virtually all heterozygous SNVs, we show how the MEC problem can serve as a framework to combine Strand-seq data with short-read, long-read, or linked-read sequencing data. Previously, Strand-seq data had only been used on its own, resulting in global yet sparse haplotypes (Porubský et al., 2016). We demonstrate how the long range phase information inherent to Strand-seq data can be leveraged to bridge phased segments obtained from Illumina, PacBio or 10X Genomics sequencing data into contiguous and global haplotypes that span whole chromosomes. We further offer extensive experimental guidance on favorable combinations of the number of used Strand-seq libraries and the depth of PacBio or Illumina coverage, and thus enable considerable reductions in costs and labor – yielding a novel, affordable and scalable approach for reconstruction of haplotype-resolved individual genomes.

4.4 Using MEC for data integration

In this section, we address the problem of haplotype phasing from multiple sequencing datasets. To this end, we present MEC instances to jointly include the read alignment or initial haplotypes from different technologies. For example, partial haplotypes might have been generated using Strand-seq data or 10x Genomics. Furthermore, the read alignments over the variants using different technologies such as PacBio or Illumina are then additionally incorporated in $\mathcal{F}$. As before, the goal is to partition the rows in $\mathcal{F}$ into two non-conflicting sets. The input matrix and the bipartition of the rows are illustrated in Figure 4.1. The matrix is filled with 0, 1 and '-' entries, where 0 and 1 indicate that the corresponding read supports the reference or alternative allele, respectively, and '-' means the information is missing (e.g. because a read does not cover this variant site). We apply the WhatsHap algorithm (Patterson et al., 2015) on the MEC instance from multiple sequencing technologies. WhatsHap selects a subset of rows and solves the wMEC problem optimally on these rows. The result is a maximum likelihood bipartition of rows, which corresponds to the two sought haplotypes. For all analyses, WhatsHap was provided with a reference genome (option –reference) to enable re-alignment-based allele detection when constructing the fragment matrix from sequencing reads. This has been shown to significantly improve performance for PacBio reads (Martin et al., 2016).

4.5 Evaluation metrics

To assess the quality of assembled haplotypes, we calculated different metrics described in the following.

- Completeness: The process of haplotyping establishes phase relations between pairs of consecutive heterozygous variants. We call each such pair a 'phase connection'. We define haplotype segment or haplotype block as a connected component of reads. For each haplotype segment produced by a combination of technologies, we therefore count the number of phase connections, which is equal to the number of heterozygous markers in the haplotype segment minus one. To measure the completeness of a phasing, we sum the number of phase connections across all haplotype segments and divide by the maximum possible number of phase connections, which is equal to the number of heterozygous variants in the chromosome minus one.
- Switch error rate: The switch error rate is the fraction of phase connections for which the phasing between the two involved heterozygous variants is wrong (see Figure 4.2).
- Largest haplotype segment: We are interested in haplotypes that span the whole length of the chromosomes. To measure extend to which we achieved this, we report the fraction of heterozygous variants that are part of the largest haplotype segment.

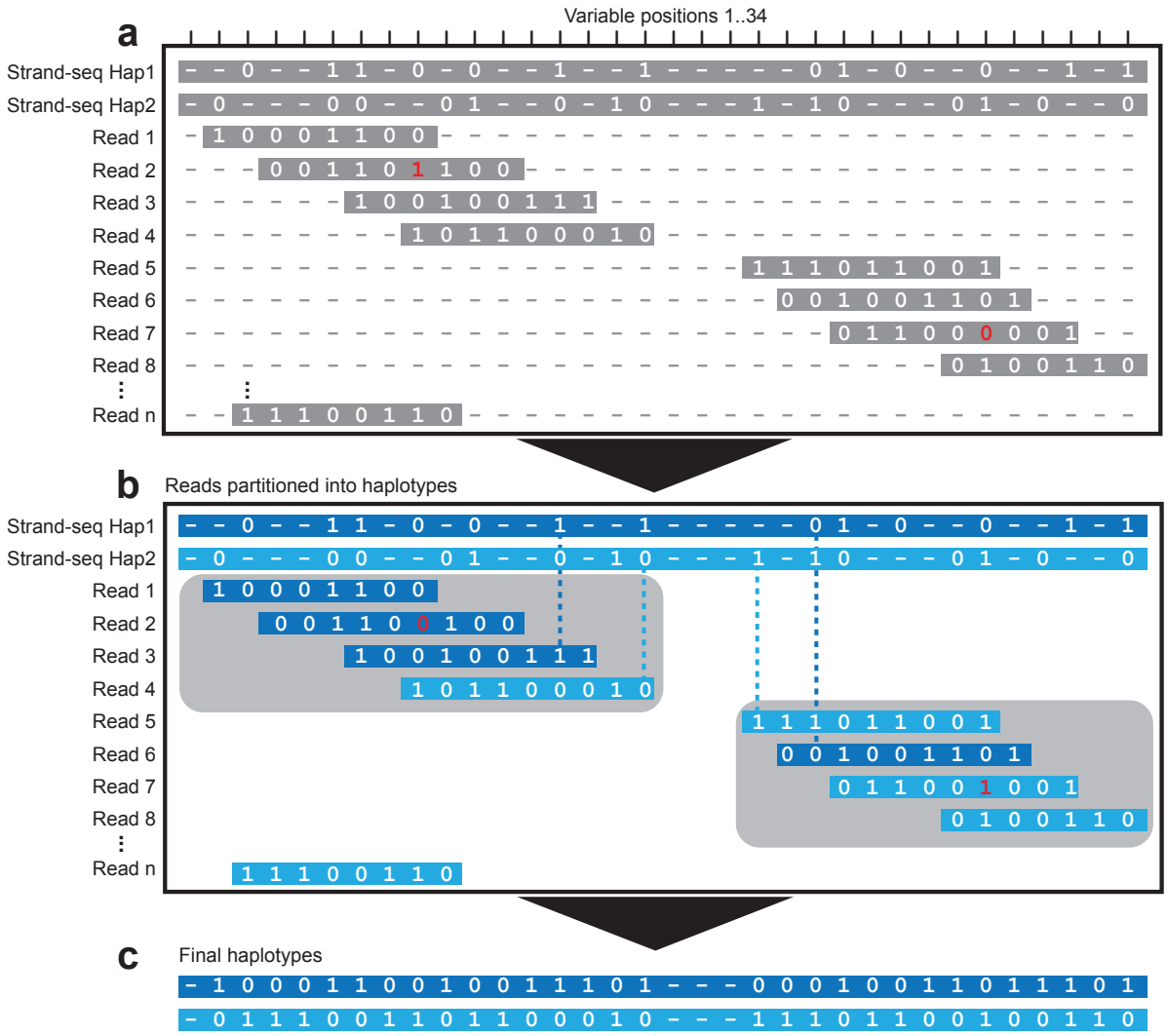

Figure 4.1: Integration of global and local haplotypes by the WhatsHap algorithm. An example solution of the weighted minimal error correction problem (wMEC) using WhatsHap algorithm is shown. For simplicity base qualities used as weights are omitted from the picture. (a)The columns of the matrix represent 34 heterozygous variants (SNVs). Continuous stretches of zeros and ones indicate alleles supported by respective reads (0 – reference allele, 1 – alternative allele). First two rows of the wMEC matrix are represented by Strand-seq haplotypes, illustrated as one 'super read' connecting alleles along the whole length of the chromosome. (1st row haplotype 1 alleles, 2nd row haplotype 2 alleles). Subsequent rows of the matrix are represented by reads that map to the reference assembly in short overlapping segments. Sequencing errors (shown in red in read 2 and 7) are corrected when the cost for flipping the alleles is minimized. (b) Reads are then partitioned into two haplotype groups (Haplotype 1 – dark blue, Haplotype 2 – light blue) such that a minimal number of alleles are corrected (in red). As an illustration of long haplotype contiguity facilitated by Strand-seq 'super reads', we depict two non-overlapping groups of reads (gray rectangles) that can be stitched together by Strand-seq (dashed lines). (c) Final haplotypes are exported for both groups of optimally partitioned reads.

- Largest haplotype segment Hamming rate: To assess whether haplotypes are correct over long genomic distances, we only consider the largest haplotype segment and compute the Hamming distance between true and predicted haplotypes (see Figure 4.2), divided by the total number of heterozygous variants in this haplotype segment. That is, the Hamming error rate is equal to the fraction of wrongly phased heterozygous variants. Note that, a single switch error (e.g. in the middle

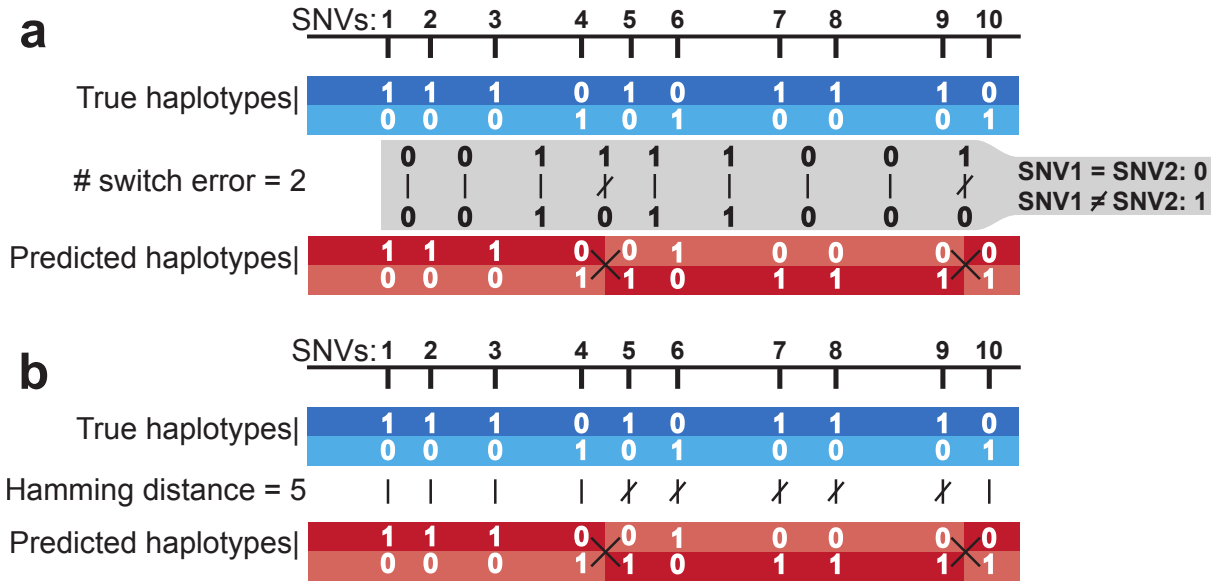

Figure 4.2: Hypothetical phasing of 10 single nucleotide variants (SNVs) along a defined chromosomal region is shown here. Each heterozygous SNV is represented in its two allelic forms (0 - reference allele, 1 - alternative allele). True (reference) haplotypes are distinguished in blue colors and predicted haplotypes in red. a) To count the number of switch errors (black crosses) between the true and predicted haplotypes, neighboring pairs of SNVs are compared along each haplotype and recorded as a new binary string of 0's and 1's depending on whether the allele state changes (see gray box). A zero value is assigned if the given pair of SNVs have the same value, otherwise a value of 1 is assigned value 1. The absolute number of differences in the binary string generated for the true and predicted haplotypes is reported as the total number of switch errors. b) To calculate the Hamming distance, the absolute number of differences between reference and predicted haplotypes is calculated for all SNV positions. In addition we calculate block-wise Hamming distance which represents a cumulative sum of all Hamming distances across all phased segments

of a chromosome) can give rise to a very large Hamming distance and hence the Hamming distance is a much more stringent quality measure. While the switch error rate assesses whether haplotypes are correct locally, i.e. between pairs of neighboring heterozygous variants, the Hamming distance assesses whether haplotypes are correct globally.

4.6 Results

4.6.1 Experimental design and dataset description

To explore a new integrative phasing strategy, with the aim of obtaining dense and accurate chromosome-length haplotypes, we used sequencing data available for a well-studied individual (NA12878). The NA12878 genome has been extensively sequenced using multiple technologies, providing high-coverage public sources of sequence information. In this study, we focused on read-based phasing data generated from Illumina short-read sequencing and PacBio technology, as they represent current standards for short- and long-read sequencing, respectively (Illumina short-read sequencing is for simplicity referred to as "Illumina data"). The Illumina dataset was sequenced to an average depth of 49.5x coverage with a median insert size of 433bp, and the PacBio dataset was sequenced to 39.6x coverage with an average read length of 15kb. In addition, we evaluated the performance of 10X Genomics, an emerging linked-read technology. Since none of these technologies alone provides chromosome-length haplotype information, we additionally incorporated single cell Strand-seq data (Porubský et al., 2016), which has

the capacity to scaffold haplotype fragments obtained from other data types (Figure 4.3(a)). Here we used 134 single cell libraries sequenced to an average depth of 0.037x coverage per library using a paired-end sequencing protocol. To evaluate the phasing accuracy of haplotypes reported in this study, we used the publicly available Illumina platinum haplotypes generated for the same individual (NA12878) as a 'reference' standard. NA12878 'reference haplotypes' were completed by genetic haplotyping using highly accurate genotypes from seventeen individuals of a three-generation pedigree (Eberle et al., 2017), which renders it an ideal gold-standard set for haplotype comparisons. We confirmed that sites and genotypes are in very good agreement with Genome in a Bottle calls. However, it should be noted that, due to stemming from short reads, this SNV set most likely lacks some variants at repetitive or complex genomic loci (e.g. recent segmental duplications).

4.6.2 Datasets

Illumina reads (Sudmant et al., 2015; Consortium et al., 2015) were obtained from the 1000 Genome Project Consortium [2]. PacBio reads (Zook et al., 2014) were obtained from Genome in a Bottle Consortium (GIAB) [3]. 10X Genomics haplotypes: Pre-assembled 10X Genomics haplotypes (produced on the Chromium platform with Chromium Genome v1 reagents, sequenced on an Illumina HiSeq X Ten and processed with LongRanger 2.1.0) were downloaded from 10X Genomics website [4] and filtered for heterozygous and PASS filter SNVs. Strand-seq libraries (Porubský et al., 2016): We downloaded them from the European Nucleotide Archive [5], accession number: PRJEB14185. The same data can also be obtained at the Zenodo site [6]. Reference haplotypes (Eberle et al., 2017): In this study we use as a gold standard, we downloaded reference triopedigree- based haplotypes of NA12878 obtained released as part of the Illumina platinum genomes (Version: 2016-1.0 from 6 June 2016) [7].

4.6.2.1 Downsampling of sequencing datasets

To assess different combinations of Strand-seq libraries (w.r.t. number of single cell libraries) with read data (w.r.t. depth of coverage), we performed a systematic analysis of the phasing performance for various subsets of each dataset. To achieve this, we downsampled the original publicly available datasets consisting of: 134 single cell Strand-seq libraries (Porubský et al., 2016), 39.6x coverage long-read PacBio data (Zook et al., 2014), and 49.6x coverage short-read Illumina data (Sudmant et al., 2015; Consortium et al., 2015). To simulate Strand-seq datasets consisting of reduced numbers of single cells, we randomly selected subsets of either 5, 10, 20, 40, 60, 80, 100, or 120 libraries from the original number of 134 libraries in the dataset. Read data from the PacBio and Illumina datasets were downsampled using Picard (picard-tools-1.130) to meet a defined depth of coverage of either 2, 3, 5, 10, 15, 25, or 30-fold. The downsampling was performed for 5 independent trials to account for variability in downsampled datasets, and the average phasing performance across all trials was reported (as described below).

4.6.3 Phasing performance of individual technologies

To independently assess the phasing performance of each technology we assembled haplotypes directly from sequencing reads (Illumina or PacBio) using WhatsHap. As discussed above (Section 5.4), this algorithm is that it solves the Minimum Error Correction (MEC) problem optimally with a run-time that scales linearly in the number of variants (alleles) and is independent of the read length. Therefore, it performs well with short-read technologies (Illumina) and is especially suited for use with long reads (PacBio, Oxford Nanopore). 10X Genomics haplotype segments were assembled by the vendor using the

[2] ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/data/NA12878/high_coverage_alignment/

[3] ftp://ftptrace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/NA12878_PacBio_MtSinai/sorted_final_merged.bam

[4] https://support.10Xgenomics.com/genome-exome/datasets/NA12878_WGS_210

[5] http://www.ebi.ac.uk/ena

[6] doi:10.5281/zenodo.830278

[7] http://www.illumina.com/platinumgenomes/

10X LongRanger pipeline. To phase multiple Strand-seq libraries we used the R package StrandPhaseR[8] developed by David Porubsky. The haplotypes generated by each technology (i.e. Illumina, PacBio, 10X Genomics and Strand-seq) were compared to the Illumina platinum reference haplotypes, to establish the density, completeness and accuracy of the phase blocks delivered by each platform independently. For a more streamlined exposition, we focus on the results obtained for Chromosome 1 in the following analysis and present numbers aggregated across all chromosomes in a concluding discussion.

We found both PacBio and 10X Genomics technologies capable of phasing nearly the complete set of variants listed in the reference haplotypes (98.8% and 97.2%, respectively), whereas Illumina alone phased only 77.8% and Strand-seq only 57.6% of the reference SNVs (Figure 4.3(b)). Note that for 10X Genomics data, we used the variant set discovered, genotyped, and phased by the 10X LongRanger software and hence variants not discovered decrease our estimate of completeness. The comparatively low percentage for Strand-seq can be explained by the relatively low sequencing coverage employed, combined with a slight unevenness in genomic coverage. For all technologies except Strand-seq, only short-range haplotypes were assembled using the read-based phasing, with a limited number of alleles phased per haplotype segment (Figure 4.3(c)). For instance, we found >30,000 unconnected haplotype segments assembled from Illumina data, with the largest segment of 16kb (median 500bp) harboring only 0.06% of the phased variants. This is because heterozygous variants that are further apart than the length of the sequenced DNA fragments cannot be connected, resulting in multiple disjoint haplotype segments with an unknown phase between them. Improvements were achieved using longer sequencing reads from PacBio technology, which effectively decreased the number of phased haplotype segments (1,927) and increased their size; the largest segment of 1.7Mb (median 21kb) containing 1.25% of all SNVs on Chromosome 1 (Figure 4.3(c)). 10X Genomics produced even longer haplotype segments than both Illumina and PacBio data (Figure 4.3(c)). The largest haplotype segment contained almost 5% of the heterozygous SNVs and spanned more than 8.5Mb (median 241kb). Still, the haplotypes of Chromosome 1 came in 199 disconnected segments and, hence, an end-to-end phasing was not achieved (Figure 4.3(c)). That is, the linked reads from the 10X Genomics were not able to connect distant neighboring heterozygous sites, for instance at centromeres, genome assembly gaps or regions of low heterozygosity (Figure 4.3(a)). This is in contrast to the global, albeit sparse, haplotypes produced by Strand-seq. Although the completeness of Strand-seq haplotypes was lower compared to the other technologies, all phased variants were placed into a single haplotype segment spanning the entire length of Chromosome 1 (Figure 4.3(b), and (c)).

Finally, we assessed the accuracy of each technology by calculating the extent of switch errors in comparison to the reference haplotypes. High phasing accuracy of each technology was exemplified by the low percentage (<0.4%) of switch errors (Figure 4.3(d)) with PacBio and 10X Genomics being the most accurate. Since no single phasing technology was sufficient to generate both global and dense haplotypes, we explored integrative phasing approaches that combine global, sparse haplotyping as afforded by Strand-seq technology with local high-density haplotypes from read-based phasing.

4.6.4 Integrative global phasing performance

We found that the combination of Strand-seq haplotypes with any of the other data types markedly increased the number of variants that were phased in the largest haplotype segment, albeit to differing degrees (Figure 4.4(a)). Specifically, for the Illumina data we observed the completeness of each haplotype increased gradually with the number of Strand-seq libraries used in the experiment, whereas the depth of coverage of Illumina data had only a minor but noticeable effect (Figure 4.4(a)). In contrast, the PacBio data showed a significant improvement in haplotype completeness at 10-fold genomic coverage, regardless of the number of Strand-seq libraries used (Figure 4.4(a), black arrowhead). Similar results were seen when we combined Strand-seq with the 10X Genomics haplotypes (Figure 4.4(a)). In all cases, integration of Strand-seq phasing drastically improved the contiguity of the haplotype spanning Chromosome 1 (Figure 4.4(b)). When combining Illumina data with 40 Strand-seq libraries >65% of the

[8] https://github.com/daewoooo/StrandPhaseR

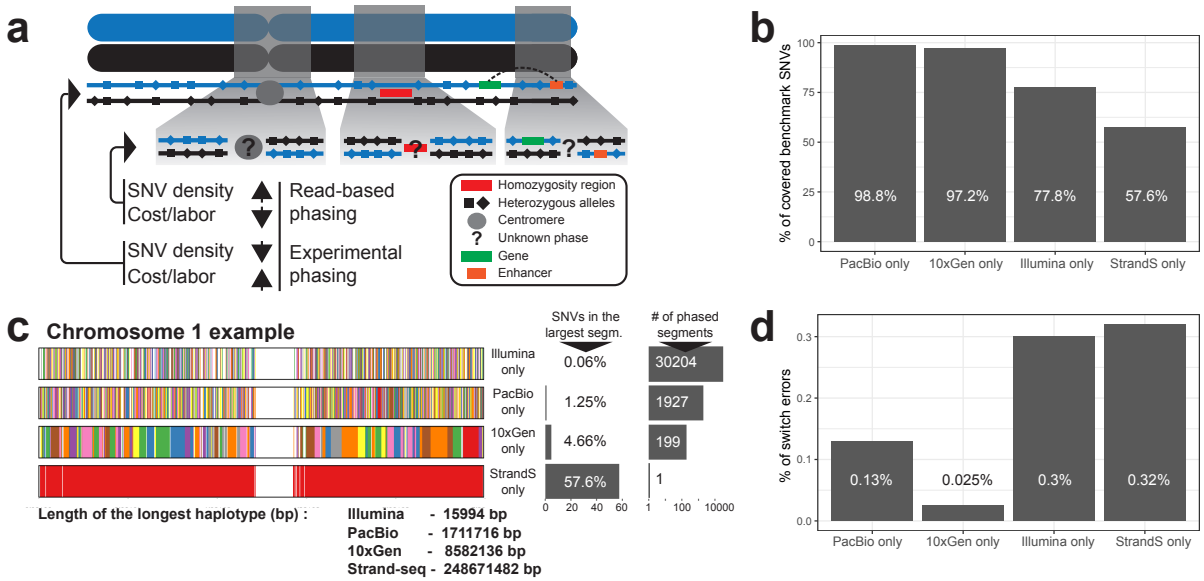

Figure 4.3: Phasing efficacy of read-based and experimental phasing approaches using Chromosome 1 as an example. a) Two homologous chromosomes are shown (blue and black). Experimental phasing approaches like Strand-seq can connect heterozygous alleles along whole chromosomes, however, at higher costs (time and labor) and lower density of captured alleles. In contrast, read-based phasing can deliver high-density haplotypes, but only short haplotype segments are assembled with an unknown phase between them. b) Barplot showing the percentage of phased variants, for each sequencing technology, from the total number of reference SNVs (Illumina platinum haplotypes). c) Graphical summary of phased haplotype segments for Illumina, PacBio, 10X Genomics and Strand-seq phasing shown for chromosome 1. Each haplotype segment is colored in a different color with the longest haplotype colored in red. Side bargraph reports the percentage of SNVs phased in the longest haplotype segment. d) Accuracy of each independent phasing approach measured as percentage of short switch errors in comparison to benchmark haplotypes.

reference variants could be phased accurately (Figure 4.4(b), black asterisk); 5497 haplotype segments (collectively representing 19.7% of the phased SNVs), however, remained disconnected, even when integrating the complete (N=134) Strand-seq dataset. These results confirm that Illumina data are of limited utility for haplotype phasing.

In contrast, as few as 10 Strand-seq cells combined with 10-fold PacBio coverage were sufficient to phase more than 95% of all heterozygous SNVs into a single haplotype segment (Figure 4.4(b), black asterisk), and merely 5 Strand-seq single cell libraries were required to connect all 10X Genomics haplotypes. However, we recommend at least 10 Strand-seq libraries (Figure 4.4(b), black asterisk) to ensure that at least one haplotype-informative (i.e. Watson-Crick-type) cell exists for every chromosome with high probability (p=0.978). This global haplotyping was unique to Strand-seq, as the combination of 10X Genomics with PacBio reads proved inefficient to join locally phased segments (Figure 4.4(b)). That is, the added value of combining these two technologies is limited as the haplotype segments tend to break at similar locations.

Finally, we assessed the phasing accuracy of the assembled haplotypes (the longest phased segment only) (Figure 4.4(c)). Similar to the completeness of the haplotype, the accuracy of Illumina phasing gradually increased with sequencing depth and Strand-seq library number, indicating that Illumina coverage of 30-fold and higher is advisable (Figure 4.4(c)). We further observed slightly elevated switch error rates at lower PacBio depths, which plateaued at 10-fold coverage (Figure 4.4(c), black arrowhead). This is likely caused by allele uncertainty resulting from error-prone PacBio reads, especially at lower sequencing depths (Figure 4.4(c)). The lowest switch error rate (< 0.2%) was achieved by the combination

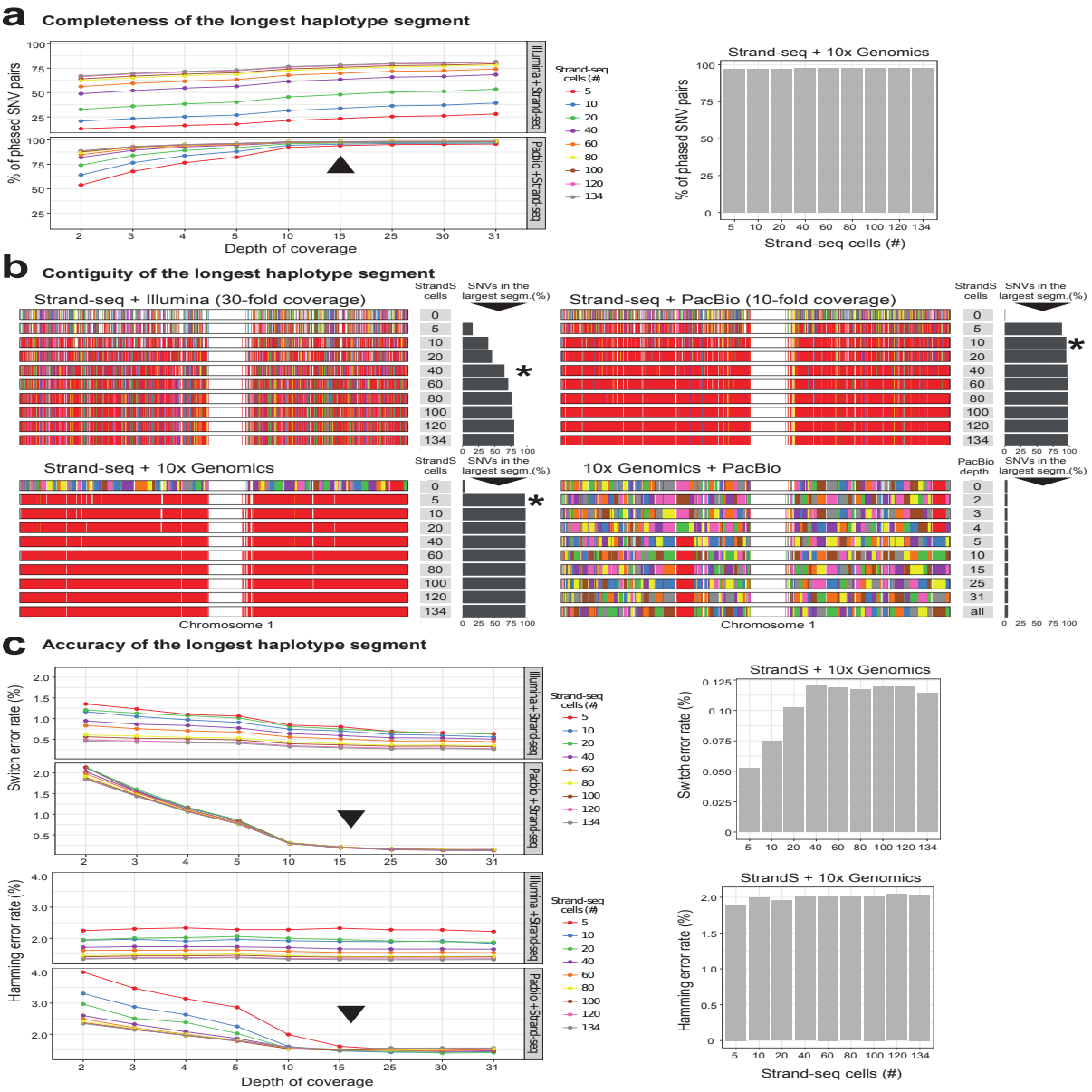

Figure 4.4: Various combinations of Strand-seq and read-based phasing (Illumina, PacBio, 10X Genomics) - Chromosome 1 as an example. Plots show haplotype quality measures for various combinations of Strand-seq cells (5, 10, 20, 40, 60, 80, 100, 120, 134) with selected coverage depths of Illumina or PacBio sequencing data (2, 3, 4, 5, 10, 15, 25, 30, >30-fold), or in combination with 10X Genomics haplotypes. a) Assessment of the completeness of the largest haplotype segment as the % of phased SNVs. Grey bars highlight PacBio sequencing depth where completeness and accuracy of final haplotypes do not dramatically improve. b) Assessment of the contiguity of the largest haplotype segment as the length of the largest haplotype segment. Every phased haplotype segment is depicted as a different color, with the largest segment colored in red. Black asterisks point to a recommended depth of coverage of a given technology in combination with Strand-seq c) Assessment of the accuracy of the largest haplotype segment as the level of agreement with the 'reference' standard. Black arrowheads highlight Illumina and PacBio sequencing depth where accuracy of final haplotypes do not substantially improve.

of Strand-seq with 10X Genomics data (Figure 4.4(c), switch error rate).

Switch error rates reflect local inaccuracies expressed by the number of pairs of consecutive heterozygous variants that are wrongly phased with respect to each other. These error rates are not necessarily informative about global haplotype accuracy, which largely depend on how switch errors are spatially distributed. Note that one single switch error implies that all following alleles (up to the next switch error) are assigned to the wrong haplotype. Since our goal is to generate dense and global haplotypes, we additionally report the Hamming error rate of the largest haplotype segment in comparison to the reference haplotypes. Illumina reads are highly accurate and therefore we observed lower impact of sequencing depth on the global accuracy of the largest phased haplotypes (Figure 4.4(c), Hamming error rate). In contrast, PacBio reads exhibited higher sequencing error rates, which translated into higher switch error rates at low sequencing depths. Using 10-fold PacBio coverage combined with at least 10 Strand-seq cells yielded highly accurate global haplotypes (Figure 4.4(c), black arrowhead), while lower coverages led to markedly worse results. Furthermore, the combination of Strand-seq with 10X Genomics haplotypes yielded highly accurate global haplotypes, already at the minimal amount of Strand-seq libraries (Figure 4.4(c), right panel).

Taken together, these results illustrate that Strand-seq can be used to phase existing sequence data and build dense, global and highly accurate haplotypes. Indeed, we found our approach highly efficient for genome-wide phasing (Figure 4.5(a)). Using a combination of 40 Strand-seq libraries with 30-fold Illumina coverage, or 10 Strand-seq libraries with either 10-fold PacBio coverage or the 10X Genomics haplotypes we successfully scaffolded chromosome-length haplotypes for every autosome of NA12878. The completeness of the genome-wide haplotypes measured for the largest haplotype block reached 95.7% and 69.1% using PacBio and Illumina reads, respectively (Figure 4.5(a)). We further demonstrated the high accuracy of these haplotypes on the local and global scales, which showed low switch (<0.45%) and Hamming error (<0.99%) rates for both the PacBio and Illumina combination (Figure 4.5(a)). Whereas scaffolding the 10X Genomic haplotypes produced the most accurate local haplotypes (switch error rate of 0.05%), global performance suffered, and the highest Hamming error rate (2.18%) was calculated for this combination. Nevertheless, using Strand-seq to scaffold any of the datasets remarkably improved the completeness, contiguity and accuracy of phasing for each chromosome, highlighting our integrative phasing strategy as a robust method for building dense and accurate whole genome haplotypes.

4.7 Discussion

Strand-seq has been successfully prepared from a wide range of cell types taken from various organisms (Porubský et al., 2016; Falconer et al., 2012; Sanders et al., 2017) and is currently being adopted by an increasing number of researchers. The integrative phasing strategy, which is a parameterized algorithm, paves the way to leveraging Strand-seq to obtain chromosome-length dense and accurate haplotypes at a manageable cost and labor investment. Based on the comprehensive evaluation presented above, we recommend three different combinations of Strand-seq with a complementary technology (Figure 4.5(b)).

As one option, one can combine Strand-seq with standard Illumina sequencing. Although the power of Illumina data for phasing is limited, mainly due to short insert sizes and read lengths, it still has some merit for adding additional variants to Strand-seq haplotypes. This might be of interest to many researchers since Illumina sequencing still constitutes the most common technology and there is an abundance of Illumina sequence data currently available for many sample genomes. To completely phase these preexisting data, we recommend generating at least 40 Strand-seq libraries for the sample genome, which is sufficient to phase >68% of all heterozygous variants genome-wide with good accuracy (switch error 0.45%, Hamming error 0.99%), see Figure 4.5(a).

To build more complete haplotypes, we recommend combining Strand-seq with either PacBio or 10X Genomic technologies. A minimum of 10-fold PacBio coverage coupled with 10 Strand-seq libraries will phase >95% of heterozygous variants genome-wide with excellent accuracy (switch error 0.25%, Hamming error 0.91%). PacBio has been demonstrated to be particularly powerful for resolving structural variation (Huddleston et al., 2017; Chaisson et al., 2015) and, although not explored here, might

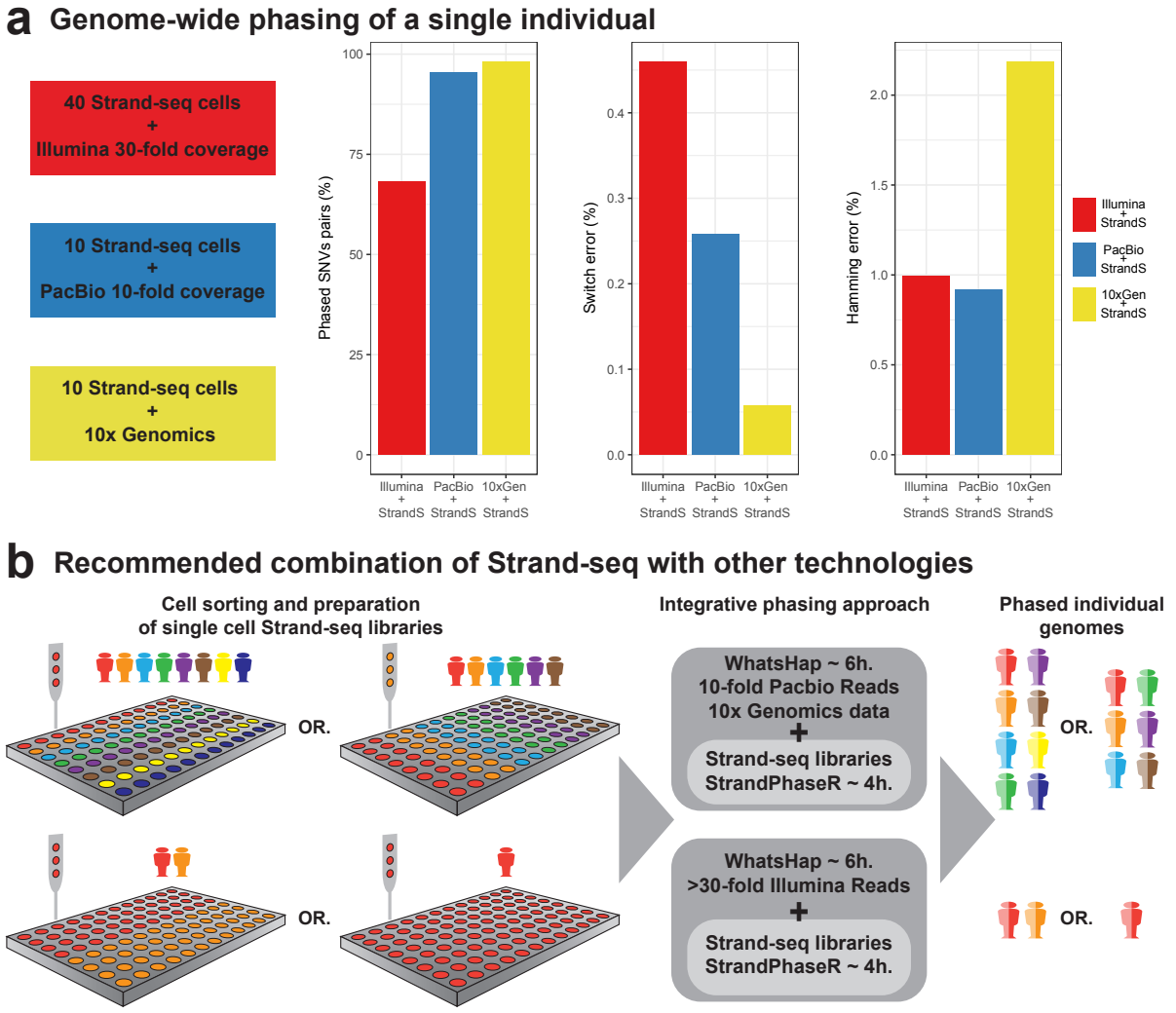

Figure 4.5: Recommended settings to phase certain amounts of individuals. (a) Genome-wide phasing of NA12878 using combination of 40 Strand-seq libraries with 30× short Illumina reads, 10 Strand-seq libraries with 10-fold long PacBio reads, or 10 Strand-seq libraries with 10X Genomics data. Plots show quality measures such as percentage of phased SNV pairs, switch error rate, and Hamming error rate for phased autosomal chromosomes. (b) A diagram providing the recommendations for the required number of Strand-seq libraries to be combined with recommended minimum of 10-fold PacBio and 30× Illumina coverage in order to reach global and accurate haplotypes for a depicted number of individual diploid genomes.

hence be the best choice when the resolution of haplotypes, structural variation and repetitive regions is desired. However, the cost of this platform is still comparatively high. Therefore, until long-read technologies have become standard practice, we recommend combining 10 Strand-seq libraries with 10X Genomics technology. We found this combination yielded the most complete (>98% heterozygous variants genome-wide) haplotypes with the lowest switch error rate (0.05%). We did observe a slightly increased Hamming error rate (2.18%), however, which indicates that some genomic intervals are placed on the wrong haplotype, most likely due to switch errors in the pre-phased haplotype segments (produced by 10X Genomics) used as input. Overall, combining Strand-seq with 10X Genomics is the most cost-effective (in terms of time and money) strategy to phase an individual genome at extraordinary accuracy.

In this study, we used pre-phased 10X Genomics haplotype segments because using the raw sparse

linked read data leads to algorithmically challenging wMEC problem instances, which presently cannot be solved optimally by WhatsHap. This implies that variants that have not been discovered by LongRanger are considered unphased (and hence decrease "completeness") and that the error rates can likely be improved further by solving the combined instance resulting from Strand-seq and 10X data. We therefore consider processing the 10X Genomics raw data an important topic of future research.

In this chapter, we focused on single individual haplotyping to avoid the biases and limitations of reference-panel based phasing as well as the need to have access to genetic material of the parents. In cases when high-coverage sequencing data of the parents are available, such datasets can be used to enhance read-based phasing and provide long range phase information.

In the next chapter, we will analyze how we can generalize this parameterized algorithm to incorporate trio information to performing phasing. This will provide possibilities to generate good quality haplotypes for pedigrees, which will have profound implications to the study of variability of personal genomes in health and disease.

This chapter has been published in Nature Communications, 2017 (Porubsky et al., 2017), which has co-authors as David Porubsky, Ashley Sanders, Jan O. Korbel, Victor Guryev, Peter M. Lansdorp & Tobias Marschall. The preliminary version of this paper is presented in David's thesis. The figures in this chapter are taken from the paper. My contribution involves in developing and testing an integrative phasing analysis pipeline. This analysis pipeline outputs data table, which is used for the figures. I also implemented the features in WhatsHap to include Strand-seq haps and 10X Genomics data. I wrote draft for WhatsHap section and the corresponding draft figures.

Chapter **5**

Parameterized algorithm for phasing pedigrees

Read-based phasing deduces the haplotypes of an individual from sequencing reads that cover multiple variants, while genetic phasing takes only genotypes as input and applies the rules of Mendelian inheritance to infer haplotypes within a pedigree of individuals. Combining both into an approach that uses these two independent sources of information – reads and pedigree – has the potential to deliver results better than each individually.

In this chapter, we provide a theoretical framework combining read-based phasing with genetic haplotyping, and describe a fixed-parameter algorithm and its implementation for finding an optimal solution. We show that leveraging reads of related individuals jointly in this way yields more phased variants and at a higher accuracy than when phased separately, both in simulated and real data. Coverages as low as 2× for each member of a trio yield haplotypes that are as accurate as when analyzed separately at 15× coverage per individual.

5.1 Introduction

With sequencing cost decreasing at an exponential rate, it has now become affordable[1] to sequence the genomes of multiple related individuals in a pedigree. As a result, sequencing datasets from pedigrees are becoming publicly available. Thus designing algorithms that solve the haplotyping problem for pedigrees in a joint framework by considering both information sources, sequencing reads and principles of Mendelian inheritance, is very important.

We recall that phasing methods can be classified into three classes. First, haplotypes can be inferred from genotype information of large cohorts based on the idea that common ancestry gives rise to shared haplotype tracts, as reviewed by Browning and Browning (2011); Loh et al. (2016b,a). This approach is known as *statistical* or *population-based phasing*. The idea is to explain the genotypes of the target genome by finding maximum likelihood paths through a reference panel (large set of haplotypes from the population). It can be applied to unrelated individuals and only requires genotype data, which can be measured at low cost. While very powerful for common variants, this technique is less accurate for phasing rare variants and cannot be applied at all to private or *de novo* variants. Second, haplotypes can be determined based on genotype data of related individuals, known as *genetic haplotyping* (Glusman et al., 2014). To solve the phasing problem, one seeks to explain the observed genotypes under the constraints imposed by the Mendelian laws of inheritance, while being parsimonious in terms of recombination events. For larger pedigrees, such as parents with many children, this approach yields highly accurate haplotypes (Roach et al., 2011; Abecasis et al., 2002; Williams et al., 2010). On the other hand, it is less accurate for single mother-father-child trios and has the intrinsic limitation of not being able to phase variants that are heterozygous in all individuals. Third, the sequences of the two haplotypes can be determined experimentally, called *molecular haplotyping*. Many techniques do not resolve the full-length haplotypes but yield blocks of varying sizes. Approaches furthermore

[1] https://www.genome.gov/27565109/the-cost-of-sequencing-a-human-genome/

largely differ in the amount of work, DNA, and money they require. As discussed in Chapter 1, on one end of the scale, next-generation sequencing (NGS) instruments generate local phase information of the length of a sequenced fragment at ever-decreasing costs. On the other end, upcoming long-read technologies such as Pacific Biosciences and Oxford Nanopore technologies produce long reads in the order of magnitude of kilo-bases. For reviews of several computational approaches that utilize data from these technologies to produce haplotypes (Rhee et al., 2016; Sedlazeck et al., 2018). As discussed in Chapter 2, WhatsHap is an efficient parameterized approach for phasing a single individual. It can, for instance, use long read data from PacBio technology and solve the Minimum Error Correction problem to produce two haplotypes.

Hybrid Approaches. The ideas underlying population-based phasing, genetic haplotyping, and read-based phasing have been combined in many ways to create hybrid methods. Delaneau et al. (2013a), for instance, use local phase information provided by sequencing reads to enhance their population-based phasing approach SHAPEIT. Exploiting pedigree information for statistical phasing has also been demonstrated to significantly improve the inferred haplotypes (Marchini et al., 2006; Chen et al., 2013). Using their heuristic read-based phasing approach HapCompass, Aguiar and Istrail (2013) note that combining reads from parent-offspring duos increases performance in regions that are identical by descent (IBD). Beyond this approach, we are not aware of prior work to leverage family information towards read-based phasing.

Contributions. Here, we build upon WhatsHap (Patterson et al., 2014, 2015) and generalize it to jointly handle sequencing reads of related individuals. To this end, we define the Weighted Minimum Error-Correction on Pedigrees Problem, termed PedMEC, which generalizes the (weighted) MEC problem and accounts for Mendelian inheritance and recombination. This problem is NP-hard. We generalize the WhatsHap algorithm for solving this problem optimally and thereby show that PedMEC is fixed-parameter tractable. When the maximum coverage is bounded, the run-time of our algorithm is linear in the number of variants and does not explicitly depend on the read length, hence inheriting the favorable properties of WhatsHap.

We target an application scenario where related individuals are sequenced using error-prone long-read technologies such as PacBio sequencing. As a driving question motivating this research, we ask how much coverage is needed for resolving haplotypes in related individuals as opposed to single or unrelated individuals. Our focus is on phasing and we do not consider the genotyping step, which can either be done from the same data or from orthogonal and potentially cheaper data sources such as micro-arrays or short-read sequencing. On simulated and real PacBio data, we show that sequencing each individual in a mother-father-child trio to $5\times$ coverage is sufficient to establish a high-quality phasing. This is in stark contrast to state-of-the-art single-individual read-based phasing, which yields worse results even for $15\times$ coverage with respect to both error rates and numbers of phased variants. We furthermore demonstrate that our technique also exhibits favorable properties of genetic haplotyping approaches: Because of genotype relationships between related individuals, we are able to infer correct phases even *between* haplotype blocks that are *not connected* by any sequencing reads in any of the individuals.

5.2 The Weighted Minimum Error Correction Problem on Pedigrees

As discussed in Chapter 1, read-based phasing has predominantly been formulated as the Minimum Error Correction (MEC) problem (Cilibrasi et al., 2007) and its weighted sibling wMEC (Greenberg et al., 2004).

In this section, we present a novel formulation for jointly phasing individuals in a pedigree. To this end, we generalize wMEC (see Problem 1.2) to account for multiple individuals in a pedigree simultaneously while modeling inheritance and recombination. An overview of notation we use is provided in Table 5.1. We assume our pedigree to contain a set of N individuals $\mathcal{I} = \{1, \ldots, N\}$.

Table 5.1: Overview of common notation.

Notation	Meaning	Example
$\mathcal{I}$	Set of individuals	$\{1, 2, 3, 4\}$
$\mathcal{T}$	Set of trio relationships	$\{(1, 2, 3), (1, 2, 4)\}$
$\mathcal{F}_i \in \{0, 1, -\}^{R_i \times M}$	Input SNP matrix for individual $i \in \mathcal{I}$	$\begin{bmatrix} - & - & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & - \end{bmatrix}$
$\mathcal{W}_i \in \mathbb{N}^{R_i \times M}$	Matrix of weights for individual $i \in \mathcal{I}$	$\begin{bmatrix} 0 & 0 & 10 & 21 & 7 \\ 13 & 9 & 31 & 25 & 0 \end{bmatrix}$
$\mathcal{X} \in \mathbb{N}^M$	Recombination cost vector	$(5, 20, 12, 23, 11)$
$g_i \in \{0, 1, 2\}^M$	Input genotypes for individual i	$(0, 2, 2, 1, 1)$
$A(k)$	Set of reads active in column k	$\{[- - 1\ 0\ 1], [0\ 1\ 1\ 1\ -]\}$
$\Delta_C(k, B, t)$	Local cost for column k, bipartition B, and transmission tuple t	10
$C(k, B, t)$	DP table entry for column k, bipartition B, and transmission tuple t	37
$h_i^0, h_i^1 \in \{0, 1\}^M$	Sought haplotypes for individual i	$(0, 1, 1, 1, 0), (0, 1, 1, 0, 1)$
$t_{m\to c}, t_{f\to c} \in \{0, 1\}^M$	Sought transmission vectors for trio $(m, f, c) \in \mathcal{T}$	$(0, 0, 0, 1, 1)$

Relationships between individuals are given as a set of (ordered) mother-father-child triples $\mathcal{T}$. For example, if $\mathcal{I} = \{1, 2, 3, 4\}$, then $\mathcal{T} = \{(1, 2, 3), (1, 2, 4)\}$ corresponds to a pedigree where individuals 1 and 2 are the parents of individuals 3 and 4. We only consider non-degenerate cases without circular relationships and where each individual appears as a child in at most one triple. Furthermore, we assume all considered variants to be non-overlapping and bi-allelic. Each individual i comes with a *genotype vector* $g_i \in \{0, 1, 2\}^M$, giving the genotypes of all M variants. Genotypes 0, 1, and 2 correspond to being homozygous in the reference allele, heterozygous, and homozygous in the alternative allele, respectively. In the context of phasing, we can restrict ourselves to the set of variants that are heterozygous in at least one of the individuals, that is, to variants k such that $g_i(k) = 1$ for at least one individual $i \in \mathcal{I}$. For each individual $i \in \mathcal{I}$, a number of R_i aligned sequencing reads is provided as input, giving rise to one SNP matrix $\mathcal{F}_i \in \{0, 1, -\}^{R_i \times M}$ and one weight matrix $\mathcal{W}_i \in \mathbb{N}^{R_i \times M}$ per individual. We seek to compute two haplotypes $h_i^0, h_i^1 \in \{0, 1\}^M$ for all individuals $i \in \mathcal{I}$. As before in the MEC problem, we want these haplotypes to be consistent with the sequencing reads.

In addition, we want the haplotypes to respect the constraints given by the pedigree. Recall that in each parent, the two homologous chromosomes recombine during meiosis to give rise to a haploid gamete that is passed on to the offspring. Therefore, each haplotype of a child should be representable as a mosaic of the two haplotypes of the respective parent with few recombination events. To control the number of recombination events, we assume a per-site recombination cost of $\mathcal{X}(k)$ to be provided as input. Controlling the recombination cost per site is important because it is not equally likely to happen at all points along a chromosome. Instead, *recombination hotspots* exist, where recombination is much more likely to occur (and should hence be penalized less strongly in our model). The cost $\mathcal{X}(k)$ should be interpreted as the (phred-scaled) probability that a recombination event occurs between variant $k-1$ and variant k. To formalize the inheritance process, we define *transmission vectors* $t_{m\to c}, t_{f\to c} \in \{0, 1\}^M$ for each triple $(m, f, c) \in T$. The values $t_{m\to c}(k)$ and $t_{f\to c}(k)$ tell which allele at site k is transmitted by mother and father, respectively. The haplotypes we seek to compute have to be *compatible* with transmission vectors, defined formally as follows.

Definition 5.1 (Transmission vector compatibility). For a given trio $(m, f, c) \in \mathcal{T}$, the haplotypes

$h_m^0, h_m^1, h_f^0, h_f^1, h_c^0, h_c^1 \in \{0,1\}^M$ are *compatible* with the transmission vectors $t_{m\to c}, t_{f\to c} \in \{0,1\}^M$ if

$$h_c^0(k) = \begin{cases} h_m^0(k) & \text{if} \quad t_{m\to c}(k) = 0 \\ h_m^1(k) & \text{if} \quad t_{m\to c}(k) = 1 \end{cases}$$

and

$$h_c^1(k) = \begin{cases} h_f^0(k) & \text{if} \quad t_{f\to c}(k) = 0 \\ h_f^1(k) & \text{if} \quad t_{f\to c}(k) = 1 \end{cases}$$

for all $k \in \{1, \ldots, M\}$.

With this notion of transmission vectors, recombination events are characterized by changes in the transmission vector, that is, by positions k with $t_{m\to c}(k-1) \neq t_{m\to c}(k)$ or $t_{f\to c}(k-1) \neq t_{f\to c}(k)$. Given our recombination cost vector $\mathcal{X}$, the cost associated to a transmission vector can be written as follows (in slight abuse of notation).

Definition 5.2 (Transmission cost). For a transmission vector $t_{p\to c} \in \{0,1\}^M$ with $p \in \{m, f\}$ and a recombination cost vector $\mathcal{X} \in \mathbb{N}^M$, the *cost* of $t_{p\to c}$ is defined as

$$\mathcal{X}(t_{p\to c}) := \sum_{k=2}^{M} [\![t_{p\to c}(k-1) \neq t_{p\to c}(k)]\!] \cdot \mathcal{X}(k),$$

where $[\![S]\!] = 1$ if statement S is true and 0 otherwise.

To state the problem of jointly phasing all individuals in $\mathcal{I}$ formally, it is instrumental to consider the set of matrix entries to be flipped explicitly. We will therefore introduce a set of index pairs $E_i \subset \{1, \ldots, R_i\} \times \{1, \ldots, M\}$ where $(j,k) \in E_i$ if and only if the bit in row j and column k of matrix $\mathcal{F}_i$ is to be flipped.

Problem 5.1 (Weighted Minimum Error Correction on Pedigrees, PedMEC). Let a set of individuals $\mathcal{I} = \{1, \ldots, N\}$, a set of relationships $\mathcal{T}$ on $\mathcal{I}$, recombination costs $\mathcal{X} \in \mathbb{N}^M$, and, for each individual $i \in I$, a sequencing read matrix $\mathcal{F}_i \in \{0, 1, -\}^{R_i \times M}$ and corresponding weights $\mathcal{W}_i \in \mathbb{N}^{R_i \times M}$ be given. Determine a set of matrix entries to be flipped $E_i \subset \{1, \ldots, R_i\} \times \{1, \ldots, M\}$ to make $\mathcal{F}_i$ feasible and two corresponding haplotypes $h_i^0, h_i^1 \in \{0,1\}^M$ for each individual $i \in \mathcal{I}$ as well as two transmission vectors $t_{m\to c}, t_{f\to c} \in \{0,1\}^M$ for each trio $(m, f, c) \in \mathcal{T}$ such that

$$\sum_{i\in\mathcal{I}} \sum_{(j,k)\in E_i} \mathcal{W}_i(j,k) + \sum_{(m,f,c)\in\mathcal{T}} \mathcal{X}(t_{m\to c}) + \mathcal{X}(t_{f\to c})$$

takes a minimum, subject to the constraints that all haplotypes are compatible with the corresponding transmission vectors, if existing.

Note that for the special case of $\mathcal{I} = \{1\}$ and $\mathcal{T} = \emptyset$, PedMEC is identical to wMEC. Therefore, the PedMEC problem is also NP-hard. As discussed in Section 5.1, we are specifically interested in an application scenario were the genotypes are already known. By using genotype data, we aim to most beneficially combine the merits of genetic haplotyping and read-based haplotyping. We therefore extend the PedMEC problem to incorporate genotypes and term the resulting problem PedMEC-G.

Problem 5.2 (PedMEC with genotypes, PedMEC-G). Let the same input be given as for Problem 5.1 (PedMEC) and, additionally, a genotype vector $g_i \in \{0, 1, 2\}^M$ for each individual $i \in \mathcal{I}$. Solve the PedMEC problem under the additional constraints that $h_i^0 + h_i^1 = g_i$ for all $i \in \mathcal{I}$, where "+" refers to a component-wise addition of vectors.

For the classical MEC problem, additionally assuming that all sites to be phased are heterozygous is common (Chen et al., 2013). This variant of the MEC problem is a special case of PedMEC-G with $\mathcal{I} = \{1\}$ and $\mathcal{T} = \emptyset$ and $g_1(k) = 1$ for all k.

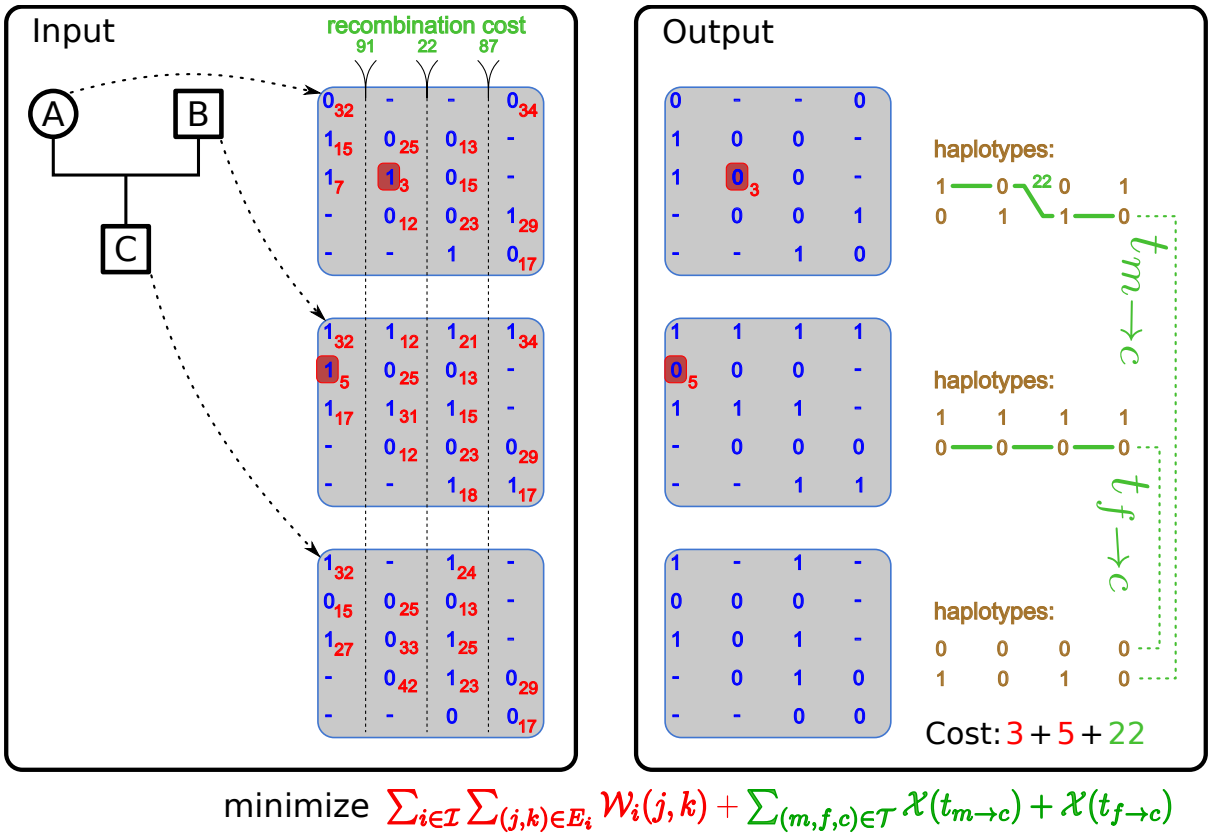

Figure 5.1: Example shows the input instance and cheapest solution and the resultant haplotypes.

5.3 Example of PedMEC

In the following, we will see how the ideas of WhatsHap algorithm can be extended for solving PedMEC and PedMEC-G. The basic idea is to use the same technique on the union of the sets of active reads across all individuals $i \in \mathcal{I}$, while adding some extra book-keeping to satisfy the additional constraints imposed by pedigree and genotypes.

We consider the example in Figure 5.1 to illustrate how to solve a PedMEC instance. Consider a trio with three individuals, mother, father and child. As input, the sequencing reads of each individual are represented as SNP matrices $\mathcal{F}_i$ for $i \in \mathcal{I} = \{A, B, C\}$. Also, we are given corresponding weight matrices $\mathcal{W}_i$ that represent the likelihood of sequencing errors in each entry of the fragment matrices. Also, we have a recombination vector $\mathcal{X}$, representing the likelihood of recombination between two consecutive variants. The sequencing errors present in reads create conflicts in these matrices. The objective here is to generate conflict-free matrices such that we obey the Mendelian laws of inheritance. This objective is achieved by jointly minimizing the flipping cost of the selected set of entries for all individuals and the cost of recombination events in mother, father or both. As an output, we obtain two haplotypes for each individual.

In this small example, it is easy to see that the entries marked in red boxes create conflict in matrices. The cheapest solution is to flip these bits by paying a cost of 3 and 5, plus allowing one recombination event in mother for a cost of 22.

Let us see how we can solve this PedMEC instance using a dynamic programming approach. We assume that there are two haplotypes marked in green and purple for the mother, and in brown and blue for the father as shown in Figure 5.2. The haplotypes for the child can be determined based on the haplotypes transmitted from mother and father.

In our DP, we go column-wise from left to right. Let us consider the first column and compute

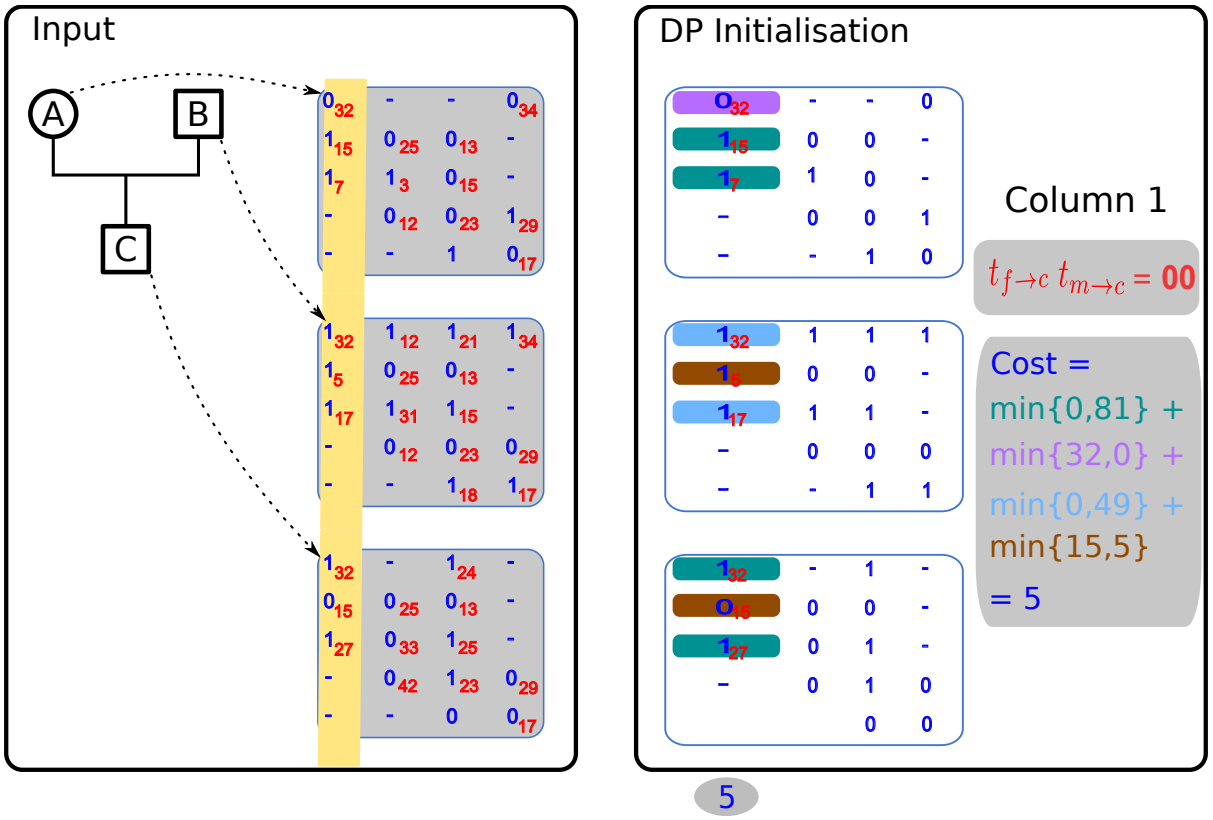

Figure 5.2: Example showing bipartition cost for the transmission vector 00 at column one.

DP cell value for the partitions shown in Figure 5.2 and a transmission value of 00; where the two digits indicate which haplotype is transmitted from mother to child and father to child respectively. Here, a value of 00 means that the mother transmits the green haplotype and the father transmits the brown haplotype. To compute the minimum-cost allele assignment for each partition, we try different possible allele assignments. For example, first, we flip all the entries in the partition to 1; and pay costs based on the weights of the corresponding entries and, second, we try to flip entries to 0 and pay costs accordingly. For the green partition, we pay a cost of 0 if we flip all entries to 0 and otherwise 15+7+32+27 = 81 for flipping all entries to 1. Similarly, we compute the allele assignment costs for other partitions too. We further take the minimum allele assignment for each partition and add the costs from all partitions, which results in cost 5 for this example. Let us consider a different transmission value 01. For this transmission value, the purple haplotype is transmitted from mother to child. Accordingly, the partitions change and their corresponding allele assignment costs change correspondingly. For the green partition, which is now transmitted to the child, costs of 0 if we flip all bits to 0 and otherwise we obtain costs 22 for flipping all bits to 1. As before, we compute the minimum allele assignment cost for each partition and resultant cost is 37 for this example. Similarly, we can compute partitions cost for all possible partitions by considering different transmission value and store them in a column of our DP table.

Let us compute the partition cost for column two shown in Figure 5.3, given the DP column for column one. Figure 5.3 shows partitions for transmission value 00, we compute the initialization cost as we computed before in column one. For example, for green partition, we compute the initialization cost (=3) as before, but we now additionally consider different possibilities of recombination events between two consecutive columns. Therefore, we pay an additional cost of 91 if the considered DP cell at column one has transmission values 01 or 10, by allowing one recombination event for both cases. In this way, when we compute the DP cell cost, we try all possibilities of recombination events (00, 01, 10,

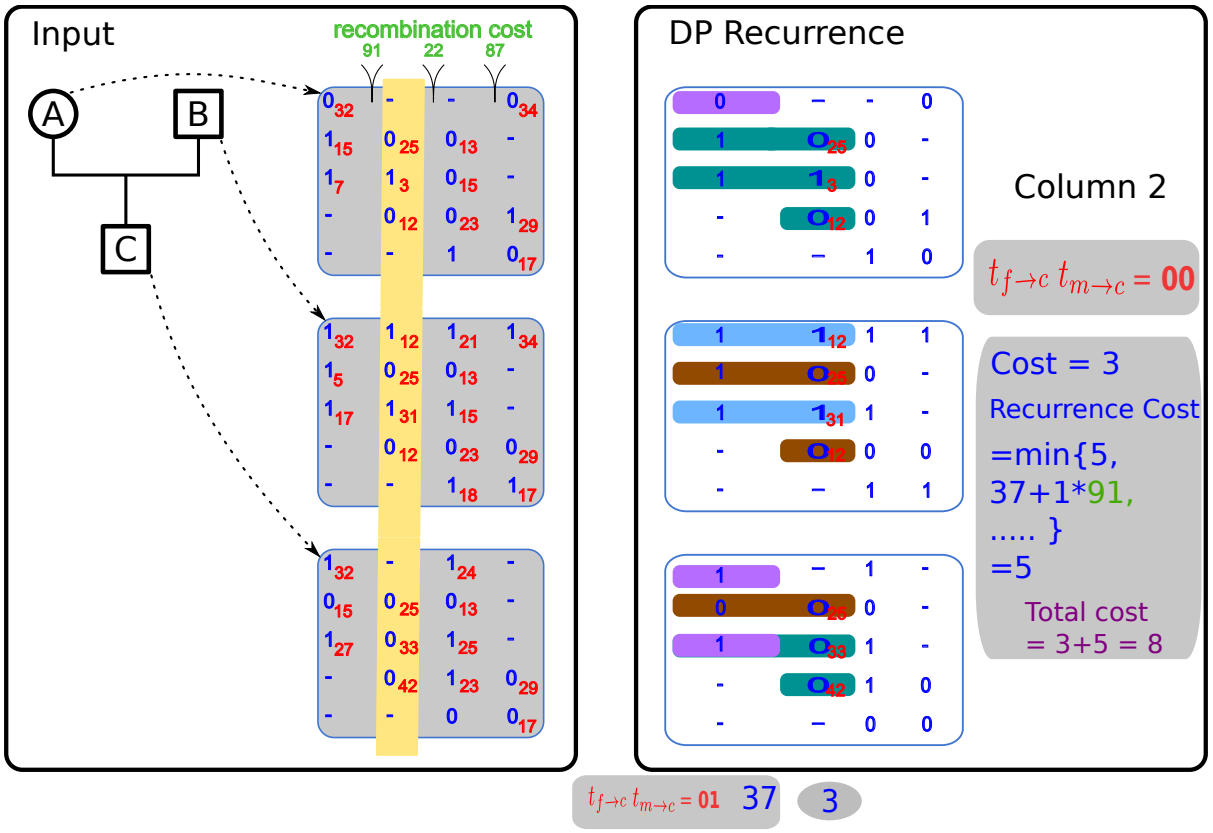

Figure 5.3: Example showing bipartition cost for the transmission vector 00 at column two, given DP column one.

11) with the previous cells and then take the minimum cost. Additionally, we consider the recursion cost from the column one such that the partitions are consistent. For this example partition, it is easy to see that the total cost is 8. Similarly, we compute partition cost for other partitions by trying all transmissions values and store them in DP column two.

We recurse this process until the last column. Once we know the optimal partitions at the last column, we can finally backtrace to get the haplotypes for each individual.

Next, we describe the full algorithm more formally.

5.4 Algorithm

Algorithm Overview: Solving PedMEC and PedMEC-G. Let $A_i(k)$ be the set of active reads in column k of $\mathcal{F}_i$. We now define $A(k) = \bigcup_{i\in\mathcal{I}} A_i(k)$. A bipartition $B = (P, Q)$ of $A(k)$ now induces bipartitions for each individual: $B_i = \big(P \cap A_i(k), Q \cap A_i(k)\big)$.

As before, we consider all bipartitions of $A(k)$ for each column k, but now additionally distinguish between all possible transmission values. We assume the set of trio relationships $\mathcal{T}$ to be (arbitrarily) ordered and use a tuple $t \in \{0,1\}^{2|\mathcal{T}|}$ to specify an assignment of transmission values. Such an assignment t can later (during backtracing) be translated into the sought transmission vectors: Assuming t to be an optimal such tuple at column k, its relation to the transmission vectors is given by

$$t = \big(t_{m_1\to c_1}(k), t_{f_1\to c_1}(k), t_{m_2\to c_2}(k), t_{f_2\to c_2}(k), \dots\big).$$

The transmission tuples give rise to one additional dimension of our DP table for PedMEC(-G), as compared to the DP table for wMEC. For each column k, we compute table entries $C(k, B, t)$ for all

$2^{|A(k)|}$ bipartitions of reads and all $2^{2|\mathcal{T}|}$ possible transmission tuples, for a total of $2^{|A(k)|+2|\mathcal{T}|}$ entries in this column.

Computing Local Costs. Along the lines of Patterson et al. (2015), we first describe how to compute the cost incurred by flipping matrix entries in each column, denoted by $\Delta_C(k, B, t)$, and then explain how to combine them with entries in $C(k-1, \cdot, \cdot)$ to compute the cost $C(k, B, t)$. The crucial point for dealing with reads from multiple individuals in a pedigree is to realize that matrix entries from haplotypes that are identical by descent (IBD) need to be identical (or need to be flipped to achieve this). For unrelated individuals (i.e. $\mathcal{T} = \emptyset$), none of the haplotypes are IBD, giving rise to $2|\mathcal{I}|$ sets of reads for the $2|\mathcal{I}|$ unrelated haplotypes. These $2|\mathcal{I}|$ sets of reads are given by B and the cost $\Delta_C(k, B, t)$ can be computed by flipping all matrix entries of reads within the same set to the same value.

For a non-empty $\mathcal{T}$, the transmission tuple t tells which parent haplotypes are passed on to which child. In other words, t identifies each child haplotype to be IBD to a specific parent haplotype. We can therefore merge the corresponding sets of reads since all reads coming from haplotypes that are IBD need to show the same allele and need to be flipped accordingly. In total, we obtain $2|\mathcal{I}| - 2|\mathcal{T}|$ sets of reads, since each trio relationship implies merging two pairs of sets. We write $\mathcal{S}(k, B, t)$ to denote this set of sets of reads induced by bipartition B and transmission tuple t in column k. The cost $W^a_{k,S}$ of flipping all entries in a read set $S \in \mathcal{S}(k, B, t)$ to the same allele $a \in \{0, 1\}$ is given by

$$W^a_{k,S} = \sum_{(i,j)\in S} [\![\mathcal{F}_i(j, k) \neq a]\!] \cdot \mathcal{W}_i(j, k),$$

where we identify reads in S by a tuple (i, j), telling that it came from individual i and corresponds to row j in $\mathcal{F}_i$. For PedMEC, i.e. if no constraints on genotypes are present, every set S can potentially be flipped to any allele $a \in \{0, 1\}$. Hence, the cost is given by

$$\Delta_C(k, B, t) = \min_{a\in\{0,1\}^{\mathcal{S}(k,B,t)}} \left\{ \sum_{S\in\mathcal{S}(k,B,t)} W^{a(S)}_{k,S} \right\}, \tag{5.1}$$

that is, we minimize the sum of costs incurred by each set of reads $S \in \mathcal{S}(k, B, t)$ over all possible assignments of alleles to read sets. For PedMEC-G, this minimization is constrained to only consider allele assignments consistent with the given genotypes. To ensure that valid assignments exist, we assume the input genotypes to be free of Mendelian conflicts.

Computing a Column of Local Costs. To compute the whole column $\Delta_C(k, \cdot, \cdot)$, we proceed as follows. In an outer loop, we enumerate all $2^{2|\mathcal{T}|}$ values of the transmission tuple t. For each value of t, we perform the following steps: We start with bi-partition $B = (A(k), \emptyset)$ and compute all $W^a_{k,S}$ for all sets $S \in \mathcal{S}(k, B, t)$ and all $a \in \{0, 1\}$, which can be done in $\mathcal{O}(|A(k)| + |\mathcal{I}|)$ time. Next we enumerate all bipartitions in Gray code order, as done previously (Patterson et al., 2015). This ensures that only one read is moved from one set to another in each step, facilitating constant time updates of the values $W^a_{k,S}$. The value of $\Delta_C(k, B, t)$ is then computed from the $W^a_{k,S}$'s according to Equation (5.1), which takes $\mathcal{O}(2^{2|\mathcal{I}|} \cdot |\mathcal{I}|)$ time. Computing the whole column $\Delta_C(k, \cdot, \cdot)$ hence takes $\mathcal{O}\big(2^{2|\mathcal{T}|} \cdot (2^{|A(k)|} + 2^{2|\mathcal{I}|} \cdot |\mathcal{I}|)\big)$ time.

DP Initialization. The first column of the DP table, $C(1, \cdot, \cdot)$, is initialized by setting $C(1, B, t) := \Delta_C(1, B, t)$ for all bipartitions B and all transmission tuples t.

DP Recurrence. Recall that $C(k, B, t)$ is the cost of an optimal solution for input matrices restricted to the first k columns under the constraints that the sought bipartition extends B and that transmission happened according to t at site k. Entries in column $C(k+1, \cdot, \cdot)$ should hence add up local costs incurred in column $k + 1$ and costs from the previous column. To adhere to the semantics of $C(k + 1, B, t)$,

only entries in column k whose bipartitions are *compatible* with B are to be considered as possible "predecessors" of $C(k+1, B, t)$.

Definition 5.3 (Bipartition compatibility). Let $B = (P, Q)$ be a bipartition of A and $B' = (P', Q')$ be a bipartition of A'. We say that B and B' are *compatible*, written $B \simeq B'$, if $P \cap (A \cap A') = P' \cap (A \cap A')$ and $Q \cap (A \cap A') = Q' \cap (A \cap A')$.

Two bipartitions are therefore compatible when they agree on the intersection of the underlying sets. Besides ensuring that bipartitions are compatible, we need to incur recombination costs in case the transmission tuple t changes from k to $k+1$. Formally, entries in column $k+1$ are given by

$$C(k+1, B, t) = \Delta_C(k+1, B, t) + \min_{\substack{B' \in \mathcal{B}(A(k)) : B' \simeq B \\ t' \in \{0,1\}^{2|\mathcal{T}|}}} \{C(k, B', t') + d_H(t, t') \cdot \mathcal{X}(k+1)\}, \tag{5.2}$$

where $\mathcal{B}(A(k))$ denotes the set of all bipartitions of $A(k)$ and d_H is the Hamming distance. The distance $d_H(t, t')$ hence gives the number of changes in transmission vectors and thus the term $d_H(t, t') \cdot \mathcal{X}(k+1)$ gives the recombination cost to be added.

Projection Columns. To ease computing $C(k+1, B, t)$ via Equation (5.2), we use the same technique described by Patterson et al. (2015) and define intermediate *projection columns* $C^{\cap}(k, \cdot, \cdot)$. They can be thought of as being *between* columns k and $k+1$. Consequently, they are concerned with bipartitions of the intersection of read sets $A(k) \cap A(k+1)$ and hence contain $2^{|A(k) \cap A(k+1)| + 2|\mathcal{T}|}$ entries, which are given by

$$C^{\cap}(k, B', t) = \min_{\mathcal{B}(A(k)) : B \simeq B'} \{C(k, B, t)\}. \tag{5.3}$$

These projection columns can be created while computing $C(k, \cdot, \cdot)$ at no extra (asymptotic) run-time. Using these projection columns, Equation (5.2) becomes

$$C(k+1, B, t) = \Delta_C(k+1, B, t) + \min_{t' \in \{0,1\}^{2|\mathcal{T}|}} \{C^{\cap}(k, B \cap A(k), t') + d_H(t, t') \cdot \mathcal{X}(k+1)\}, \tag{5.4}$$

where $B \cap A(k) := (P \cap A(k), Q \cap A(k))$ for $B = (P, Q)$. We have therefore reduced the run-time of computing this minimum to $\mathcal{O}(2^{2|\mathcal{T}|})$.

Runtime. Computing one column of local costs, $\Delta_C(k, \cdot, \cdot)$, takes $\mathcal{O}\big(2^{2|\mathcal{T}|} \cdot (2^{|A(k)|} + 2^{2|\mathcal{I}|} \cdot |\mathcal{I}|)\big)$ time, as discussed above. For each entry, we use Equation (5.4) to compute the aggregate value of cost incurred in present and past columns. Over all columns, we achieve a run-time of $\mathcal{O}\big(M \cdot \big(2^{2|\mathcal{T}|+c+|\mathcal{I}|}|\mathcal{I}| + 2^{4|\mathcal{T}|+c}\big)\big)$, where $c = \max_k\{|A(k)|\}$ is the maximum coverage.

Backtracing. An optimal bipartition and transmission vectors can be obtained by recording the indexes of the table entries that gave rise to the minima in equations (5.4) and (5.3) when filling the DP table and then backtracing starting from the optimal value in the last column. Optimal haplotypes are subsequently obtained using the bipartition and transmission vectors.

5.5 Experimental Setup

To evaluate the performance of our approach, we considered both real and simulated datasets.

5.5.1 Real Data

The Genome in a Bottle Consortium (GIAB) has characterized seven individuals extensively using eleven different technologies (Zook et al., 2014). The data is publicly available. Here we consider the Ashkenazim trio, consisting of three related individuals: NA24143 (mother), NA24149 (father) and NA24385 (son). We obtained a consensus genotype call set (NIST_CallsIn2Technologies_05182015) provided by GIAB, containing variants called by two independent technologies. For our benchmark, we consider all bi-allelic SNPs on Chromosome 1 called in all three individuals, amounting to 141,256 in total, and use the provided (unphased) genotypes.

Ground Truth via Statistical Phasing. To generate a ground truth phasing for comparison, we used the population-based phasing tool SHAPEITv2-r837 (Delaneau et al., 2014) with default parameters. The program was given the 1000 Genomes reference panel[2], the corresponding genetic map[3], and the unphased genotypes as input. SNPs present in the GIAB call set but absent in the reference panel were discarded, resulting in 140,744 phased SNPs, of which 58,551 were heterozygous in mother, 57,152 in father and 48,023 in child. We refer to this set of phased SNPs as *ground truth phased variants*. We emphasize that this phasing is solely based on genotypes and does not use phase information present in the reads in any way and hence is completely independent. In the following, we refer to the original genotypes from the GIAB call set (without phase information) restricted to this set of SNPs as *ground truth unphased genotypes*, which we use as input for read-based phasing experiments described below.

PacBio Data. For each individual, we downloaded aligned Pacific Biosciences (PacBio) reads[4], which had an average coverage of 42.3× for the mother, 46.8× for the father and 60.2× for the child, respectively. The average mapped read length across mother was 8,328 bp, 8,471 bp and child 8,687 bp, for mother, father, and child, respectively. For each individual, we separately downsampled the aligned reads to obtain data sets of 2×, 3×, 4×, 5×, 10×, and 15× average coverage.

10XGenomics Data. The GemCode platform marketed by 10XGenomics uses a barcoding technique followed by pooled short-read sequencing and data analysis through a proprietary software solution to resolve haplotypes. Data from this platform is available from the GIAB project and represents phase information obtained completely independently from either statistical phasing or PacBio reads. We downloaded the corresponding files[5] for comparison purposes.

5.5.2 Simulated Data

Despite the high-quality data set provided by GIAB, we sought to complement our experiments by a simulated data set. While the population-based phasing we use as ground truth is arguably accurate due to a large reference panel and the high-quality genotype data used as input, it is not perfect. Especially variants with low allele frequency present challenges for population-based phasers.

Virtual Child. As basis for our simulation, we use the haplotypes of the two parents from our ground truth phased dataset. We generated two haplotypes of a virtual child by applying recombination and Mendelian inheritance to the four parent haplotypes. In reality, recombination events are rare: All of Chromosome 1 spans a genetic distance of approximately 292 cM, corresponding to 2.9 expected recombination events along the whole chromosome. To include more recombinations in our simulated data set, we used the same genetic map as above, but multiplied recombination rates by 10. The

[2] https://mathgen.stats.ox.ac.uk/impute/1000GP_Phase3.tgz

[3] http://www.shapeit.fr/files/genetic_map_b37.tar.gz

[4] ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/(HG002_NA24385_son|HG003_NA24149_father|HG004_NA24143_mother)/PacBio_MtSinai_NIST/MtSinai_blasr_bam_GRCh37/

[5] ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/(HG002_NA24385_son|HG003_NA24149_father|HG004_NA24143_mother)/10XGenomics

recombination sites are sampled according to the probabilities resulting from applying Haldane's mapping function to the genetic distances between two variants. In line with our expectation, we obtained 26 and 29 recombination sites for mother and father, respectively. The resulting child had 41,676 heterozygous variants.

Simulating PacBio Reads. We aimed to simulate reads that mimic the characteristics of the real PacBio data set as closely as possible. For this simulation, we incorporate the variants of each individual into the reference genome (hg19) to generate two true haplotypes for each individual in our triple. We used the PacBio-specific read simulator pbsim by Ono et al. (2013) to generate a 30× data set for Chromosome 1. The original GIAB reads were provided to pgsim as a template (via option `-sample-fastq`) to generate artificial reads with the same length profile. Next, we aligned the reads to the reference genome using BWA-MEM 0.7.12-r1039 by Li (2013) with option `-x pacbio`. As before for the real data, the aligned reads for each individual were downsampled separately to obtain data sets of 2×, 3×, 4×, 5×, 10×, and 15× average coverage.

5.5.3 Compared Methods

Our main goal is to analyze the merits of the PedMEC-G model in comparison to wMEC; in particular with respect to the coverage needed to generate a high-quality phasing. The algorithms to solve wMEC and PedMEC-G described in Section 5.4 have been implemented in the WhatsHap software package[6]. We emphasize that WhatsHap solves wMEC and PedMEC-G optimally. Since the focus of this study is on comparing these two models, we do not include other methods for single-individual haplotyping. We are not aware of other trio-aware read-based phasing approaches that PedMEC-G could be compared to additionally.

The run-time depends exponentially on the maximum coverage. Therefore we prune the input data sets to a *target maximum coverage* using the read-selection method introduced by Fischer and Marschall (2016), which is implemented as part of WhatsHap. This target coverage constitutes the only parameter of our method. For PedMEC-G, we prune the maximum coverage to 5× for each individual separately. For wMEC, we report results for 5× and 15× target coverage. The respective experiments are referred to as PedMEC-G-5, wMEC-5, and wMEC-15. For wMEC, we use the additional "all heterozygous" assumption (see Section 5.4), to also give it the advantage of being able to "trust" the genotypes, as is the case for PedMEC-G. Both wMEC and PedMEC-G were provided with the ground truth unphased genotypes for the respective data set. PedMEC-G was additionally provided with the respective genetic map (original 1000G genetic map for real data and scaled by factor 10 for simulated data).

As described above, the ground truth phased variants was generated by SHAPEIT with default parameters, implying that SHAPEIT treated the three samples as unrelated individuals. For comparison purposes, we re-ran SHAPEIT and provided it with pedigree information. We refer to the resulting phased data set as *SHAPEIT-trio*. Moreover, we ran duoHMM (v0.1.7) by O'Connell et al. (2014) on the resulting files to further improve the phasing.

5.6 Performance Metrics

We compare each phased individual to the respective ground truth haplotypes separately and only consider sites heterozygous in this individual.

Phased SNPs. For read-based phasing of a single individual (wMEC), we say that two heterozygous SNPs are directly connected if there exists a read covering both. We compute the connected components in the graph where SNPs are nodes and edges are drawn between directly connected SNPs. Each connected component is called a *block*. For read-based phasing of a trio (PedMEC), we draw an edge when two SNPs are connected by a read in any of the three individuals. In both cases, we count a SNP as

[6] https://bitbucket.org/whatshap/whatshap

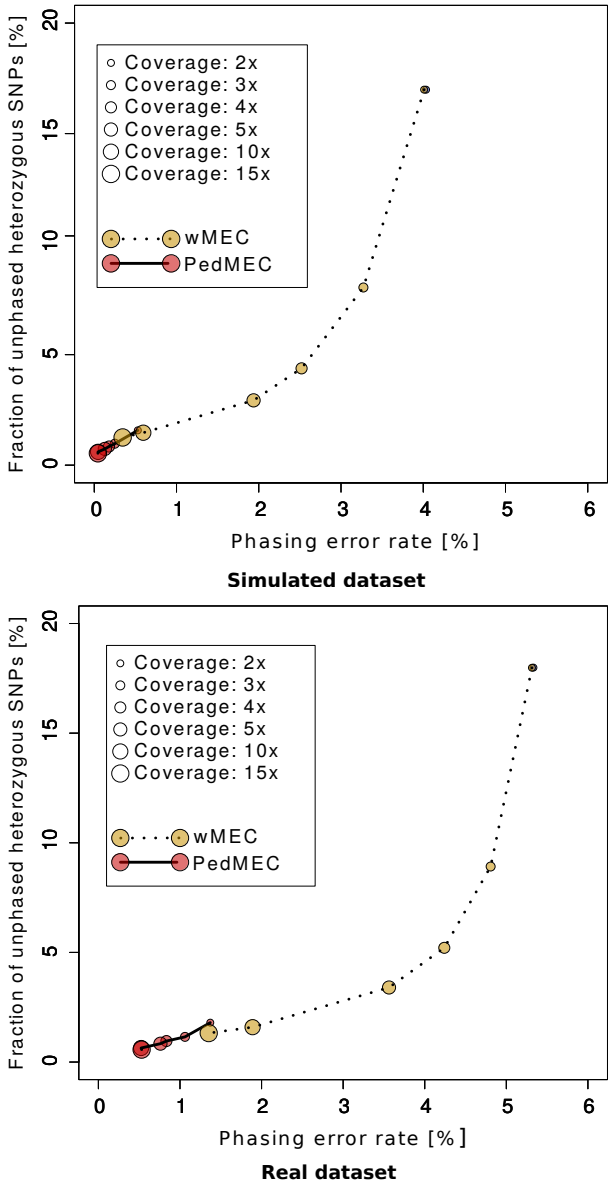

Figure 5.4: Simulated data set (top) and real dataset (bottom): phasing error rate (x-axis) versus completeness in terms of the fraction of unphased SNPs (y-axis) for PedMEC-G-5 (solid line), wMEC-5 (dashed line), and wMEC-15 (dotted line). Average coverage (per individual) of input data is encoded by circles of different sizes.

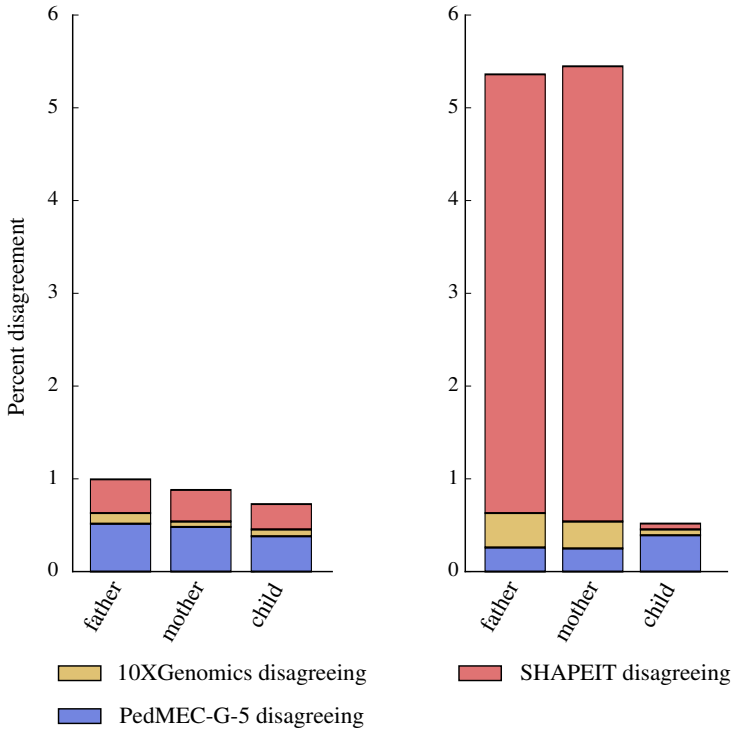

Figure 5.5: Three-way comparison of phasings provided by SHAPEIT, 10XGenomics, and PedMEC-G-5 (on 15× coverage data). Of all pairs of consecutive SNPs phased by all three methods, the percentages of cases where the phasing reported by one method disagrees with the other two are reported. Missing to 100%: cases where all three methods agree. Left: SHAPEIT run with default parameters, corresponding to our "ground truth phasing"; right: SHAPEIT run with pedigree information.

being *phased* when it is not the left-most SNP in its block (for the left-most SNP, no phase information with respect to its predecessors exists). All other SNPs are counted as *unphased*. Below, we report the average *fraction of unphased SNPs* over all three family members.

Phasing Error Rate. For each block, the first predicted haplotype is expressed as a mosaic of the two true haplotypes, minimizing the number of switches. This minimum is known as the *number of switch errors*. Note that the second predicted haplotype is exactly the complement of the first one, due to only considering heterozygous sites. When two switch errors are adjacent, they are subtracted from the number of switch errors and counted as one *flip error*. The *phasing error rate* is defined as the sum of switch and flip errors divided by the number of phased SNPs.

Three-Way Phasing Comparison. To simultaneously compare phasings from three different methods (e.g. SHAPEIT, 10XGenomics, and PedMEC-G-5), we proceeded as follows. For an individual, we considered all pairs of consecutive heterozygous SNPs that have been phased by all three methods. For each of these pairs, either all three methods agree or two methods agree (since only two possible phases exist). Below, we discuss the fraction of these different cases in relation to the total number of considered SNP pairs.

5.7 Results

We report the results of wMEC-5, wMEC-15, and PedMEC-G-5 for both data sets, *real* and *simulated*. All combinations of the three methods, two data sets, and six different average coverages (2×, 3×,

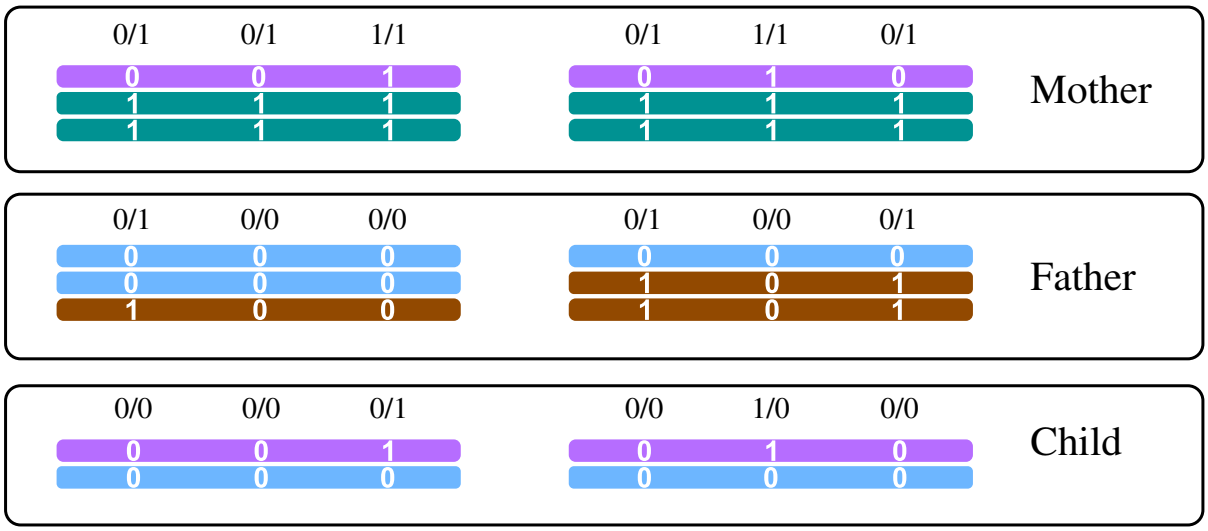

Figure 5.6: Two disjoint unconnected haplotype blocks for which phase information can be inferred from the genotypes.

4×, 5×, 10× and 15×) were run. The predicted phasings are compared to the ground truth phasing for the respective data set. That is, for the real data set, we compare to the population-based phasing produced by SHAPEIT; for the simulated data set, we compare to the true haplotypes that gave rise to the simulated reads. Figure 5.4 shows the fraction of unphased SNPs in comparison to the phasing error rate (see Section 5.6) for all conducted experiments. A perfect phasing would be located in the bottom left corner.

The Influence of Coverage. Increasing the average coverage is beneficial for phasing. For all three methods (wMEC-5, wMEC-15, and PedMEC-G-5) and both data sets, the phasing error rate and the fraction of unphased SNPs decrease monotonically when the average coverage is increased, as is clearly visible in Figure 5.4. The effect is much more drastic for wMEC than for PedMEC, however. Apparently, wMEC needs more coverage to compensate for PacBio's high error rate while PedMEC can resort to exploiting family information to resolve uncertainty.

The Value of Family Information. When operating on the same input coverage, PedMEC-G-5 clearly outperformed wMEC-5 and even wMEC-15 in all cases tested. This was true for phasing error rates as well as for the fraction of phased positions. On the real data set with average coverage 10×, for instance, wMEC-5 and wMEC-15 reached an error rate of 2.9% and 1.9%, while it was 0.5% for PedMEC-G-5.

Most remarkingly, PedMEC-G-5 delivers excellent results already for very low coverages. When working with an average coverage as low as 2× for each family member, it achieves an error rate of 1.4% and a fraction of unphased SNPs of 1.8% (on real data). In contrast, wMEC-15 needs 15× average coverage on each individual to reach similar values (1.4% error rate and 1.3% unphased SNPs). When running on 5× data, PedMEC-G-5's error rate and fraction of unphased positions decrease to 0.75% and 0.85%, respectively. Therefore, it reaches better results while requiring only a third of the sequencing data, which translates into significantly reduced sequencing costs.

Comparison of Real and Simulated Data. When comparing results for simulated and real data, i.e. top and bottom plots in Figure 5.4, the curves appear similar, with some important difference. In terms of the fraction of phased SNPs (y-axis), results are virtually identical. This indicates that our simulation pipeline establishes realistic conditions regarding this aspect. Differences in terms of error rates (x-axis) are larger. In general, error rates in the real data are larger than in the simulated data,

which might be partly caused by a too optimistic error model during read simulation. On the other hand, the population-based phasing used as ground truth for the real data set will most likely also contain errors. Especially low-frequency variants present difficulties for population-based phasers. Next, we therefore compare our ground truth statistical phasing to the independent phasing provided by 10XGenomics.

Three-Way Comparison with 10XGenomics. Figure 5.5(left) shows the results of a three-way comparison of ground truth statistical phasing, 10XGenomics, and PedMEC-G-5 on 15× coverage data. We observe that the total fraction of cases where there is disagreement between the three methods is below 1%. Out of these, 10XGenomics and the statistical phasing agree in a sizeable fraction of cases, shown in blue. In these cases, the PacBio-based PedMEC-G-5 is likely wrong. Given PacBio's high read error rate, the existence of such cases is not surprising. On the other hand, there also is a significant fraction of cases where PedMEC-G-5 and 10XGenomics agree, shown in red, indicating likely errors in the statistical phasing. Cases where PedMEC-G-5 and the statistical phasing agree but disagree with 10XGenomics are very rare, which is likely due to the low error rates of short-read sequencing underlying the 10XGenomics phasing and the resulting highly accurate phasing.

SHAPEIT-trio and duoHMM. Figure 5.5(right) shows the same three-way comparison, but uses the results obtained from SHAPEIT when run in trio mode. We see that this improved the phasing for the child but dramatically worsened the agreement for the parents, with more than 4% of all phased SNP pairs for which 10XGenomics and PedMEC-G-5 agreed but disagreed with SHAPEIT-trio. Running duoHMM (O'Connell et al., 2014) to improve the SHAPEIT-trio phasing did not lead to any changes, which might be related to that duoHMM is designed to be run for large cohorts of related individuals.

Phase Information Beyond Block Boundaries. Genetic phasing operates on genotypes of a pedigree, without using any sequencing reads. Figure 5.6 illustrates a case where we have two blocks that are not connected by reads in any individual. Nonetheless phase information can be inferred from the genotypes: Each block contains a SNP that is homozygous in both parents and heterozygous in the child, which immediately establishes which haplotype is maternal and which is paternal in both blocks. Note that this, in turn, also implies the phasing of the parents. By design, PedMEC-G implicitly exploits such information. To demonstrate this, we used the real data set and merged all blocks reported by PedMEC-G into one chromosome-wide block and determined the fraction of cases where phases were correctly inferred between blocks—and hence between two SNPs that are not connected by reads in any individual. This resulted in a fraction of 89.7% correctly inferred phased (averaged over all individuals and coverages; standard deviation 1.4%). Repeating the same for wMEC yielded 50.4% correctly inferred phased, as expected equalling a coin flip.

Runtimes. All experiments have been run on a server with two Intel Xeon E5-2670 CPUs (10 cores each) running at 2.5GHz. The implementation in WhatsHap is sequential, i.e. only using one CPU core. In all cases, the time spent reading the input files dominated the time spent in the phasing routine itself. Processing all three individuals of the 5× coverage real data set took 31.1 min, 31.2 min, and 26.2 min for wMEC-5, wMEC-15, and PedMEC-G-5, respectively. This time included all I/O and further processing. Of these times, 2.0 s, 4.1 s, and 101.0 s were spent in the phasing routine, respectively. For input coverage 15×, total processing took 89.3 min, 93.9 min, and 65.4 min for wMEC-5, wMEC-15, and PedMEC-G-5, respectively. Of this, 2.5 s, 149.9 s, 321.5 s were spent in the phasing routines, respectively. We conclude that the phasing algorithm presented here is well suited for handling current data sets swiftly. In the future, we plan to further optimize the implementation of I/O subroutines and provide automatic chromosome-wise parallelization of data processing.

5.8 Discussion

We have presented a unifying framework for integrated read-based and genetic haplotyping. By generalizing the WhatsHap algorithm (Patterson et al., 2015), we provide a fixed-parameter tractable method for solving the resulting NP-hard optimization problem, which we call *PedMEC*. When maximum coverage and number of individuals are bounded, the algorithm's run-time is linear in the number of phased variants and independent of the read length, making it well suited for current and future long-read sequencing data. This is mirrored by the fact that the run-time is dwarfed by the time required for reading the input files in practice. PedMEC can use any provided costs for correcting errors in reads as well as for recombination events. By using phred-scaled probabilities as costs, minimizing the cost can be interpreted as finding a maximum likelihood phasing in a statistical model incorporating Mendelian inheritance, read error correction, and recombination.

Testing the implementation on simulated and real trio data, we could show that the method is notably more accurate than phasing individuals separately, especially at low coverages. Beyond enhanced accuracy, our method is also able to phase a greater fraction of heterozygous variants compared to single-individual phasing.

Being able to phase more variants is a key benefit of the integrative approach. Whereas read-based phasing can in principle only phase variants connected by a path through the covering reads, adding pedigree information enables even phasing of variants that are not covered in all individuals since the algorithm can "fall back" to using genotype information. Figure 1.5 illustrates the increased connectivity while phasing a trio, resulting in more phased variants in practice.

Genetic haplotyping alone cannot phase variants that are heterozygous in all individuals, emphasizing the need for an integrative approach as introduced here. We demonstrate that such an approach indeed yields better result and recommend its use whenever both reads and pedigree information are available. Most remarkingly, the presented approach is able to deliver outstanding performance even for coverages as low as $2\times$ per individual, on par with performance delivered by single-individual haplotyping at $15\times$ coverage per individual.

This chapter has been published at ISMB Proceedings/Bioinformatics 2016 (Garg et al., 2016), which has co-authors as Marcel Martin & Tobias Marschall. The figures in this chapter are taken from the paper. I co-developed the algorithm with Tobias Marschall and co-wrote the paper with him.

Chapter **6**

A graph-based approach to diploid genome assembly

Constructing high-quality haplotype-resolved *de novo* assemblies of diploid genomes is important to reveal the full extent of structural variation and its role in health and disease. Current assembly approaches often collapse the two sequences into one haploid consensus sequence and, therefore, fail to capture the diploid nature of the organism under study. Thus, designing an assembler that is able to produce accurate and complete diploid assemblies, while being resource-efficient with respect to sequencing costs, is a key challenge to be addressed by the bioinformatics community.

In this chapter, we present a novel graph-based approach to diploid assembly, which integrates accurate Illumina data and long-read Pacific Biosciences (PacBio) data. We demonstrate the effectiveness of our method on a pseudo-yeast diploid genome and show that we require as little as 50× coverage Illumina data and 10× PacBio data to generate accurate and complete assemblies. Additionally, we show that our approach has the ability to detect and phase structural variants.

6.1 Introduction

In previous chapters, we have studied the problem of reconstructing haplotypes with respect to a given reference genome. Here, we consider de novo assembly where we seek to reconstruct all haplotype sequences without relying on a reference genome. As introduced in Chapter 1, the process of assembling the two genome sequences from sequencing reads in a haplotype-aware manner is known as *diploid* or *haplotype-aware genome assembly.* and the generated fragmented assemblies are called "haplotigs". However, the characteristics of next generation sequencing (NGS) reads vary depending on the technology employed. While reads from second generation platforms are short, third generation platform produce reads with very high error rates, which renders diploid genome assembly fundamentally challenging . Additional challenges inherent in the genome assembly problem include dealing with short and long genomic repeats, handling other rearrangements present in the genome, and scaling efficiently considering input size, genome size, and hardware availability.

Across the short, long, and hybrid categories (see Section 1.3), most current assemblers (Sohn and Nam, 2016; Simpson and Pop, 2015) require collapsing the two genome sequences of a diploid sample into a single haploid "consensus" sequence (or primary contig). The consensus sequence is obtained by merging together the distinct alleles at heterozygous regions into a single allele, and therefore losing important information. The resulting haploid *de novo* assembly does not represent the true characteristics of the diploid input genome.

To generate diploid assemblies for heterozygous genomes, there are two standard linear approaches; one uses haploid contig sequences (Chin et al., 2016; Pendleton et al., 2015; Seo et al., 2016; Mostovoy et al., 2016), while the other partitions the reads while using the reference genome as a backbone

(Glusman et al., 2014; Martin et al., 2016; Chaisson et al., 2017b). In both approaches, the reads are first aligned (either to the reference genome or the contigs). Second, variants such as SNVs are detected based on the aligned reads. Finally, the detected variants are phased using long reads from a same or different sequencing technology.

For both reference-guided and contig-based assembly , this third step—solving the phasing problem—can be addressed by solving the minimum error correction (MEC) optimization problem discussed in Chapter 3 and Chapter 4. There are several disadvantages to reference-guided assembly; for example, the reads are initially aligned to the reference genome and therefore the process is prone to reference bias. Also, this approach can fail to detect sequences or large structural variants that are unique to the genome being assembled. Thus, we can not find the cause of the disease, which is actually associated with this structural variant in the genome.

However, there are also several reasons why the set of sequences/contigs produced by contig-based assembly is not ideal. First, the contigs produced by haploid assemblers ignore the heterozygous variants in complex regions, opting instead to break contiguity to express even moderate complexity. Second, the contigs do not capture end-to-end information in the genome; the ordering or relationships between contigs are critical in order to generate end-to-end chromosomal-length assemblies.

Some newer diploid assembly methods include Weisenfeld et al. (2017), where 10x Genomics linked read data is used to determine the actual diploid genome sequence. Their approach is based on de Bruijn graphs and applies a series of graph simplifications, in which simple bubbles are detected and phased by using (short) reads that stem from the same (long) input molecule, which is determined through barcoding. There is also a recent study by Chin et al. (2016), who follows a linear phasing approach to generate diploid assemblies (*haplotigs*) for diploid genomes by using PacBio reads.

Contributions. We propose a graph-based approach to generate haplotype-aware assemblies of single individuals. Our contribution is twofold. First, we propose a hybrid approach to integrate accurate Illumina and long PacBio reads to generate diploid assemblies. The Illumina reads are used to generate an assembly graph that serves as a backbone for subsequent PacBio-based steps. Second, we generalize the diploid assembly problem to encompass constructing the diploid assembly directly from the underlying assembly graph. The two genome sequences can be seen as two paths over the regions of heterozygosity in the assembly graph. We make some first steps towards performing read-based phasing on graphs.

Figure 6.1 demonstrates the conceptual advantages of our graph-based approach over the Falcon Unzip method. Consider four SNVs separated by two large SVs and there are four reads spanning these variants. Falcon Unzip can not phase the central region because the reads r_3 and r_4 do not cover any SNVs and resulting in incomplete haplotigs. In contrast, graph-based approaches attempt to detect all types of SVs and phase all of them.

We demonstrate the feasibility of our approach by performing a haplotype-aware de novo assembly of a whole pseudo-diploid yeast (SK1+Y12) genome. We show that we generate accurate, contiguous, and correctly phased diploid genomes. Through analysis of different coverage levels, we demonstrate that we require only $50\times$ short-read coverage and as little as $10\times$ long-read coverage data to generate diploid assemblies. This illustrates that our hybrid strategy is a cost-effective way of generating haplotype-resolved assemblies. Finally, we show that we successfully detect and phase large structural variants.

6.2 Further related work

The assembly problem was initially formalized as the shortest common super-string problem (SCS)(Maier, 1978) to generate consensus sequence. The SCS problem turned out to be NP-hard, thus Tarhio and Ukkonen (1988) predominantly followed the greedy approach to find an approximate solution for solving the assembly problem. The basic idea of this approach is to combine the two reads that overlap

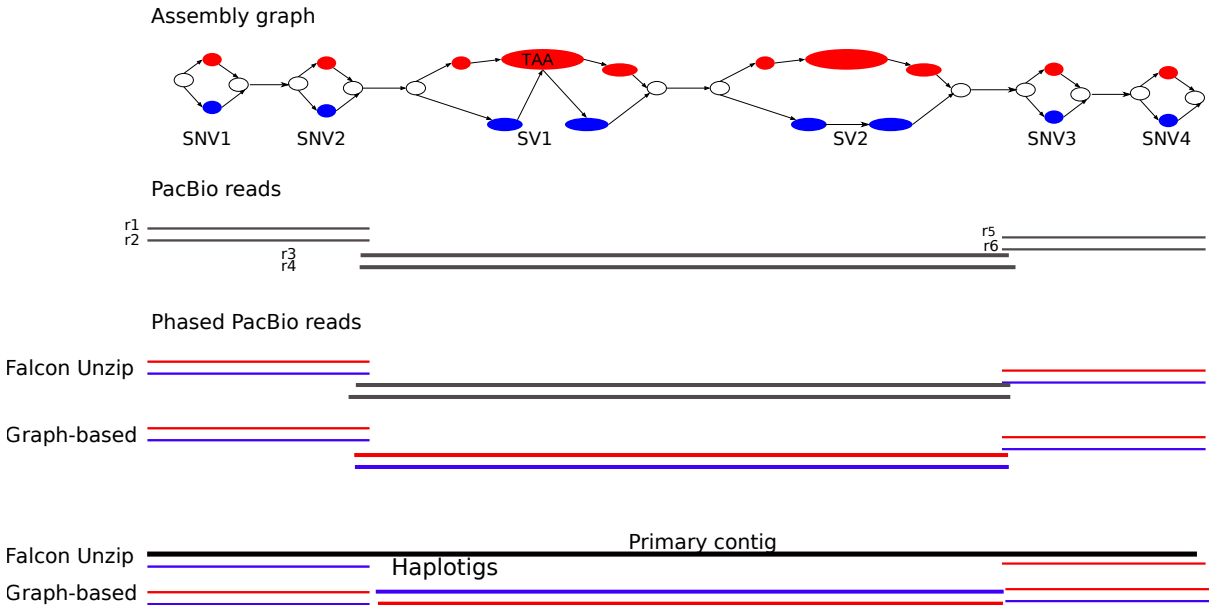

Figure 6.1: Input: an assembly graph (top) (consisting of four SNVs and two SVs) and the PacBio reads $r_1, r_2, r_3, r_4, r_5, r_6$ (gray). Output: the phased reads (colored in blue and red) and haplotigs (bottom) using Falcon Unzip and our graph-based approach. Our graph-based phases central region, contrarily, Falcon Unzip does not.

to a maximum extent in terms of length or base quality estimates. This process is repeated until a predefined minimum quality threshold is reached.

The early genome assemblers (Sutton et al., 1995; Green et al., 1999) were based on the greedy strategies and have been used during the Human Genome Project. Greedy approaches were also explored for datasets generated by second-generation DNA sequencing technologies (Warren et al., 2006; Jeck et al., 2007).

The major disadvantage of these greedy approaches is that the shortest common super-string problem neglects an essential feature of complex (e.g., vertebrate) genomes: repeats. This has led to the invention of graph-based models for sequence assembly to handle repeats and other complex genomes. Graph-based models form the basis of most current genome sequence assemblers. As introduced in Chapter 1, there are basically two types of graph models such as de Bruijn (Pevzner et al., 2001; Medvedev et al., 2007; Todd, 1933) and OLC-based string graphs (Myers, 2005). Over the last few years, a lot of effort has been devoted to improving the graph based assemblers.

The Edena assembler followed the string graph approach for short-read sequencing data (Hernandez et al., 2008). To efficiently construct the string graph, several assemblers were developed using the FM index (Simpson and Durbin, 2010), which led to memory-efficient assemblers for large genomes (Li, 2012; Simpson and Durbin, 2012). The latest version of SGA implemented Bloom filter to reduce the running time. Furthermore, several fast string graph construction algorithms were developed (Dinh and Rajasekaran, 2011; Gonnella and Kurtz, 2012; Ben-Bassat and Chor, 2014).

In the direction of de Bruijn graph, there were also considerable efforts involved. The ABySS assembler (Simpson et al., 2009) introduced a representation of the graph as a hash table of k-mers, with each k-mer storing a byte representing the presence or absence of its eight possible neighboring k-mers. This representation helped in computational gain by allowing distribution of the hash table across a cluster of computers. In recent years, the use of Bloom filters (Bloom, 1970) has been gaining popularity by representing a set of k-mers. The Bloom filter was first applied to k-mer counting by Melsted and Pritchard (2011). The FM-index data structure Ferragina and Manzini (2000), which was basically designed for mapping reads to the reference genome, was also used for sequence assembly. The FM-index has also been recently used for representing de Bruijn graphs (Bowe et al., 2012; Rødland,

2013). The key point was that only a fraction of k-mers need to be directly stored as vertices was observed by Ye et al. (2012). A similar technique was used to improve the memory consumption of the popular SOAPdenovo assembler (Luo et al., 2012). All steps in SOAPdenovo pipeline can be customized according to users requirements.

Several algorithms that particularly considered the mate-pair information during assembly have been developed . In the works of Butler et al. (2008); Bankevich et al. (2012), the assembler attempted to enumerate all the paths connecting the endpoints of a mate pair. The paired de Bruijn graph (Medvedev et al., 2011) approach modified the de Bruijn graph structure to implicitly encode the mate-pair information, thereby simplifying, resolving repeats and scaffolding of the graph based on the mate-pair information.

Long-read assemblers. Although these short-read assemblers can generate consensus sequences to span entire chromosomes, they are very fragmented and lack the finer resolution required to improve contig lengths. Instead, the biggest gains in contig lengths stem from single-molecule sequencing, which generates long-read datasets. These long-read technologies such as PacBio and ONT can generate reads longer than Illumina, but have high error rates. More specifically, these reads are long enough to cover the most common repeats in both microbial and vertebrate genomes and can therefore generate highly continuous assemblies. Subsequent efforts have been devoted to handling long read lengths and high error rates error rates to generate good quality assemblies. Several assembly tools are specifically designed for long-read PacBio and Nanopore data such as Canu (Koren et al., 2017), HINGE (Kamath et al., 2017), Racon (Vaser et al., 2017), Falcon (Chin et al., 2016), and Miniasm (Li, 2016).

Canu, which is based on Celera Assembler, has been specifically designed for noisy single-molecule sequences. Canu followed an adaptive overlapping strategy that involved aligning the PacBio reads based on tf-idf weighted MinHash (Berlin et al., 2015) and constructed a sparse assembly graph. It was mainly designed to support long-read data, while being efficient in terms of run-time and coverage requirements, and additionally performed repetition and haplotype separation. Racon corrected assemblies by finding a consensus sequence between reads and the assembly through the construction of partial order alignment (POA) graphs. After aligning the reads by mapping tools available (e.g. Minimap or Graphmap), Racon segmented the sequence and found the best alignment between a POA graph of the reads and the assembly. The HGAP (Hierarchical Genome Assembly Process) pipeline was developed for assembling PacBio-generated data without requiring correction using short-read data (Chin et al., 2013). HGAP was a hierarchical pipeline; it selected the longest PacBio reads to form the basis of the assembly and performed their error-correction using multiple sequence alignment. The corrected long reads were then assembled, and a consensus sequence was generated using the complete data set. This method often generated single-contig assemblies of bacterial genomes. The FALCON assembler followed the design of the hierarchical genome assembly process (HGAP) but used more computationally optimized components. It used string graph algorithm for genome assembly and being best suited for PacBio reads. It included several important steps such as error correction of raw reads, pre-assembly error correction, overlap filtering, graph construction from overlaps and contig construction from graph.

Hybrid assemblers. Besides short and long-read assemblers, the hybrid approaches to combine different sequencing datasets provide another avenue for generating accurate, contiguous and complete assemblies. The hybrid assemblers, SPAdes by Bankevich et al. (2012) and ALLPATHS-LG by Gnerre et al. (2011), are DBG-based, which constructs the base graph using Illumina data and then scaffold the assemblies using longer, less accurate reads such as PacBio. Furthermore, SPAdes also supports ONT data as the long read complement to NGS data. Recently, Fan et al. (2017) demonstrates that the combination of PacBio and Illumina delivers accurate structural variant calls.

Other technologies. An emerging trend to generate accurate, contiguous and complete assemblies is, to combine cost-effective Illumina sequencing dataset with other sequencing techniques. One powerful sequencing technique is chromatin conformation capture via proximity ligation and high-throughput

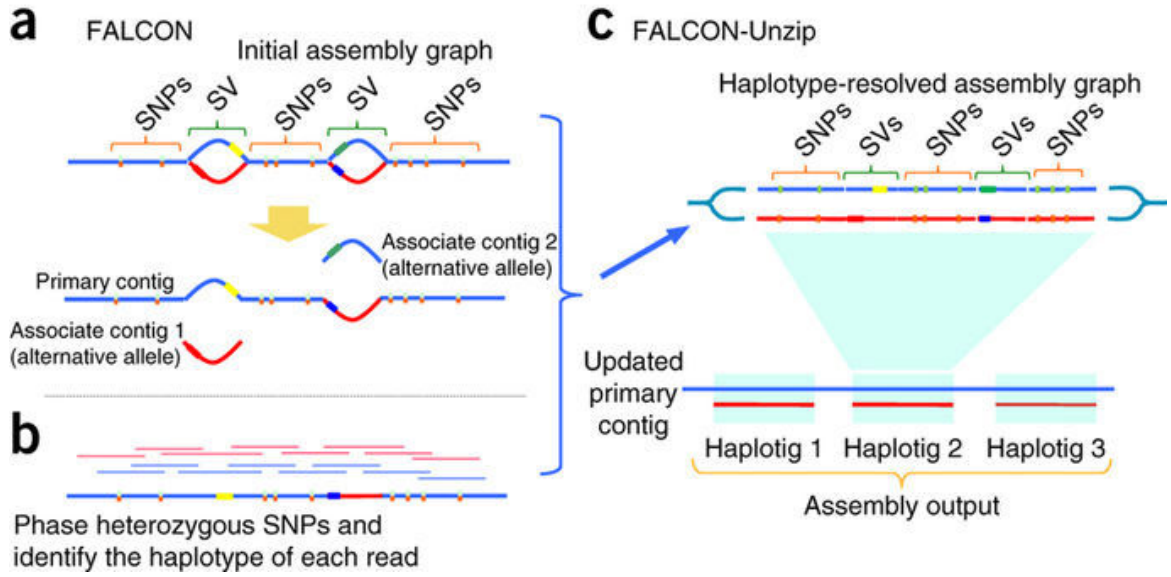

Figure 6.2: (a) An initial assembly graph is constructed by FALCON by error-correcting the reads. The bubbles are collapsed into a consensus sequence "primary contig". (b) Heterozygous SNPs are identified and phased, thus haplotype of reads is identified. (c) The phased reads are used to incorporate the haplotype-fused path into the initial assembly graph, thus finally a set of primary contigs and associated haplotigs are generated. Figure from the paper by (Chin et al., 2016).

sequencing (Hi-C) (Lieberman-Aiden et al., 2009). This technique involves in generating a paired-read data type (two reads separated by some distance) but from a distribution of sizes that can span mega bases. The main advantage of these data sets is in the scaffolding step to connect contigs, which help in generating chromosome-length scaffolds, and phase haplotypes (Burton et al., 2013; Selvaraj et al., 2013). Along the similar lines, Rice et al. (2017) demonstrates on using in vitro reconstituted chromatin and Illumina sequencing to assemble the American alligator genome. Another sequencing technique is the barcoding of short reads to tag groups of "linked reads" that all originate from a larger, single molecule of DNA. For these synthetic long-read data, Weisenfeld et al. (2017) introduces a new assembler, Supernova, for the de novo assembly of diploid human genomes. Additionally, in the latest version of the ABySS assembler, Jackman et al. (2017) explores linked reads and optical mapping for improved scaffolding.

In addition to haploid assemblers, there also exists an only available diploid assembler that has the ability to generate diploid assemblies for diploid organisms.

Diploid assemblers. To date, there is only one diploid assembler, Falcon Unzip. It's pipeline is given in Figure 6.2. Falcon Unzip first constructs a string graph by using PacBio reads and then collapses the bubbles representing divergent regions between homologous sequences (Fig. 6.2a) to generate sets of "haplotype-fused contigs" , also called "primary contigs". Next, Falcon Unzip aligns reads to the primary contigs and then identifies phases of reads using aligned reads over heterozygous SNVs. (Fig. 6.2b). Afterwards, Falcon Unzip identifies haplotypes for the genome and phases each read using greedy approach. Phased reads are then used to assemble haplotigs (Fig. 6.2c) that form the final diploid assembly with phased single-nucleotide polymorphisms (SNPs) and structural variants (SVs).

We now explain the phasing process followed by Falcon Unzip. The reads are aligned to primary contigs and heterozygous SNPs (het-SNPs) are called by considering signal from sequence alignments. A

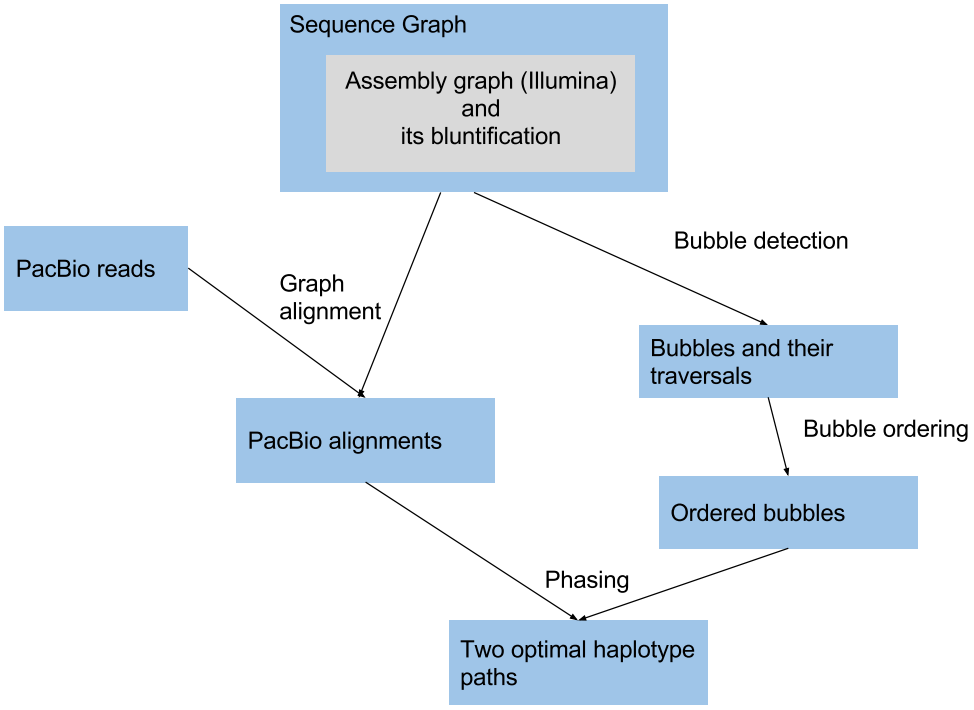

Figure 6.3: Overview of the diploid assembly pipeline.

simple phasing algorithm based on greedy approach is followed to phase "chain of SNPs". Additionally, the phase to aligned reads can be assigned unambiguously if read covers sufficient heterozygous SNVs. If a read covers sufficient het-SNPs, then Falcon Unzip assigns a phased tag to each read. Some reads might not have enough phasing information, thus do not contain phased tag. For example, if there are not enough het-SNP sites covered by a read, it assigns a special "un-phased tag" for each un-phased read. Furthermore, the initial assembly graph is updated according to phased reads and the haplotigs are generated in a greedy manner using local conservative approach.

6.3 Diploid assembly pipeline

In this section, we present a novel diploid assembly pipeline for generating diploid assemblies. Our assembly workflow uses short read (e.g. Illumina) and long read (e.g. PacBio) data combinedly, as illustrated in Figure 6.3, and we describe the details of this process in the following.

6.3.1 Sequence graph

Our first step is to construct a sequence graph using short read data with a low error rate, as provided by the Illumina platform.

Definition 6.1 (Sequence Graph). We define a sequence graph $G_s(N_s, E_s)$ as a bidirected graph, consisting of a set of nodes N_s and a set of edges E_s. The nodes n_i are sequences over an alphabet $\mathcal{A} = \{A, C, G, T\}$. For each node $n_i \in N_s$, its reverse-complement is denoted by n_i'. An edge $e_{i'j}$ connects the nodes n_i' to n_j. Nodes may be traversed in either the forward or reverse direction, with the sequence being reverse-complemented in the reverse direction.

In words, edges represent adjacencies between the sequences of the nodes they connect. Thus, the graph implicitly encodes longer sequences as the concatenated sequences of the nodes along walks through the graph.

To illustrate this, we consider an example sequence graph G_s in Figure 6.4. It consists of a node set $N_s = \{1, 1', 2, 2', 3, 3', \ldots\}$ and an edge set $E_s = \{1 \to 2, 1 \to 3' \ldots\}$.

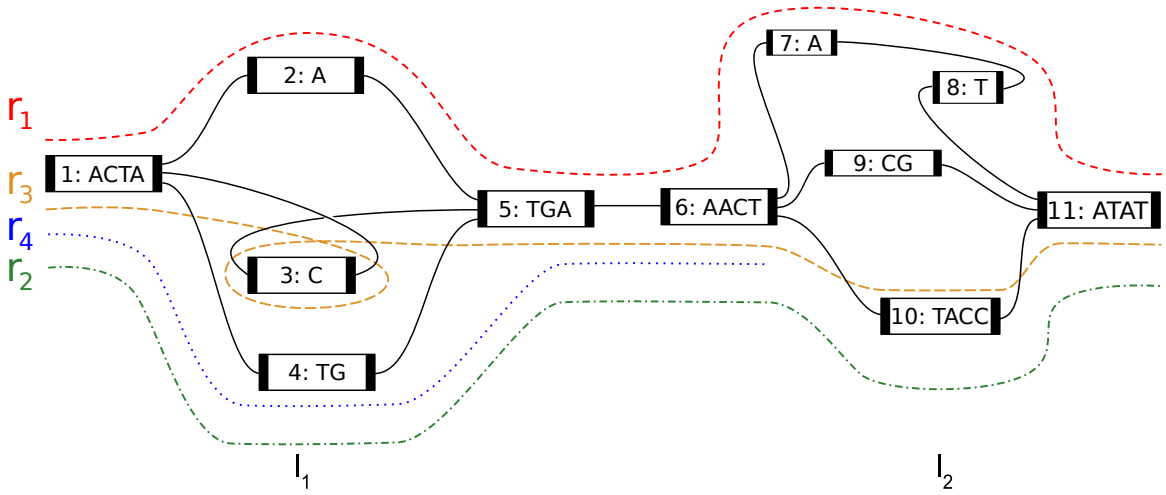

Figure 6.4: For a subgraph of G_s, the example shows two bubbles l_1 and l_2, and their corresponding alleles. Reads r_1, r_2, r_3, r_4 traverse the bubbles.

To generate the sequence graph G_s, we first employ SPAdes (Bankevich et al., 2012), which constructs and simplifies a de Bruijn graph, and we subsequently remove the overlaps between the nodes in the resulting graph in a process we call *bluntification*, explained in the Supplement.

6.3.2 Bubble detection in sequence graphs

To account for heterozygosity in a diploid genome, we perform bubble detection. The notion of *bubble* we use is closely based on the *ultrabubble* concept as defined by Paten et al. (2017). Briefly, bubbles have the following properties:

- *2-node-connectivity.* A bubble is bounded by fixed start and end nodes. Removing both the start and end nodes disconnects the bubble from the rest of the graph. Note that a bubble can be viewed in either orientation. If the graph is traversed in one direction, and a bubble is encountered that starts at a node n_i and ends at a node n_j', then that bubble can also be described as the bubble with start node n_j and end node n_i', as it would be encountered when traversing the graph in the opposite direction.
- *Directed acyclicity.* A bubble is directed and acyclic.
- *Directionality.* All paths through the bubble flow from start to end.
- *Minimality.* No vertex in the bubble other than the start node n_i (with proper orientation) forms a pair with the end node n_j' (with proper orientation) that satisfies the above properties. Similarly, no vertex in the bubble other than n_j' forms such a pair with n_i.

A bubble can represent a potential sequencing error or genetic variation within a set of homologous molecules. We represent bubbles as collections of alternative paths.

Definition 6.2 (Path). We define path a_i as a linear ordering of nodes $a_i = n_1, \ldots n_m$.

A bubble is a collection of paths with the same start and end node and can be defined as follows:

Definition 6.3 (Bubble). Formally, a bubble is represented as a collection of allele paths $l_k = \{a_1, a_2 \ldots\}$ where

$$a_1 = (n_1, n_2, \ldots n_m), a_2 = (n_1, n_3 \ldots n_m)$$

and so on.

For example, Figure 6.4 shows a set of two bubbles $L = \{l_1, l_2\}$, and the set of allele paths for the bubble l_2 is $\{a_1, a_2, a_3\}$, where $a_1 = (6, 7, 8', 11)$, $a_2 = (6, 9, 11)$, $a_3 = (6, 10, 11)$.

6.3.3 PacBio alignments

For phasing bubbles, we consider long reads from third generation sequence technologies such as PacBio. We align these long reads to the sequence graph G_s to generate paths through the graph. We perform graph alignment using a banded version of the algorithm described by Rautiainen and Marschall (2017), which is a generalization of semi-global alignment to sequence-to-graph alignment[1].

There are several advantages of aligning PacBio reads to graphs instead of to a reference genome or contigs. SNPs often occur near larger variants such as insertions and deletions. SNPs are thus often missed in these regions when reads contain large mismatches with respect to the linear sequences they are aligned against. Graph alignment allows the alignment of reads to variants appropriate to each read's phase, and to other types of complex events.

Definition 6.4 (Alignment). We define a set of read alignments as $R = \{r_1, r_2, \ldots, r_j\}$, where each read alignment r_j is given by a path of oriented nodes in graph G_s, written $r_j = (n_1, \ldots, n_m)$.

For example, in Figure 6.4, $R = \{r_1, r_2, r_3, r_4\}$ and the read alignment path r_1 can be written as $r_1 = (1, 2, 5, 6, 7, 8', 11)$.

6.3.4 Bubble ordering

The next stage of our algorithm is to obtain an ordering of the bubbles $L = (l_1, l_2, \ldots l_k)$, which we refer to as a *bubble chain*. For example, in Figure 6.4, $L = (l_1, l_2)$ is a bubble chain. A general sequence graph G_s is cyclic, due to different types of repeats present in the genome that create both short and long cycles. Ordering bubbles in such a graph is closely related to resolving repeats, which is a challenging problem. In this study, we rely on the Canu algorithm (Koren et al., 2017) to provide a bubble ordering by aligning Canu-generated contigs to our sequence graph. Furthermore, we detect repetitive bubbles—that is, bubbles that would need to be traversed more than once in a final assembly—based on the depth of coverage of aligned PacBio reads, and remove such bubbles. We deem a bubble repetitive if the number of PacBio reads aligned to its starting node is greater than a coverage threshold specified by the user over the genome. For example, given a $30\times$ ($= c$) data set and a repeat that occurs 20 ($= r$) times in the genome, then the coverage at the bubble on average is 600 ($= r \cdot c$).

6.3.5 Graph-based phasing

Given a sequence graph G_s, ordered bubbles L, and PacBio alignments R, the goal is to reconstruct two haplotype sequences $\{h_0, h_1\}$, called haplotigs, along each chain of bubbles.

Definition 6.5 (Haplotype path). Formally, a pair of haplotype paths (h_0, h_1) can be defined as two paths through a bubble chain in the sequence graph and denoted as:

$$h_0 = (n_s, n_2, \ldots n_e)$$

$$h_1 = (n_s, n_3, \ldots n_e)$$

where h_0 and h_1 may differ at the heterozygous regions defined by bubbles, and n_s and n_e are the start and end of the bubble chain.

The two genome sequences can be seen as two walks through the bubbles L in the sequence graph G_s that are consistent with the PacBio alignments R. In maximum likelihood terminology, the goal is to find the most likely haplotype paths given the alignment paths traversing through the bubbles. For example, in Figure 6.4, given bubbles (l_1, l_2) and PacBio alignments $R = \{r_1, r_2, r_3, r_4\}$, the goal is to find two maximum likelihood haplotype paths $\{h_0, h_1\}$ such that each PacBio alignment is assigned to one of the haplotypes.

[1] https://github.com/maickrau/GraphAligner

For a linear chain of bubbles L, the task of finding these two haplotype paths is equivalent to picking one allele path per haplotype for each bubble. To this end, we note that an alignment path r_j for a given read can be viewed as a sequence of allele paths traversed in consecutive bubbles. We represent this association of reads to allele paths in the form of a *bubble matrix* $\mathcal{F} \in \{0, 1, \ldots m, -\}^{|R|\times|L|}$, where $|R|$ is the number of reads, $|L|$ is the number of bubbles along a chromosome, and $m = \max_k |l_k|$ is the maximum number of paths (or alleles) in any bubble $l_k \in L$. The entry $\mathcal{F}(j, k) \in \{0, 1, \ldots m, -\}$ represents the allele path index in bubble l_k that read r_j is aligned to, where a value of "–" indicates that the read does not cover the bubble. In Figure 6.4, note that the read alignment path r_4 does not cover all the nodes in any of the allele paths in l_2 and hence we set the corresponding value $\mathcal{F}(4, 2)$ to "–". As a result, this read covers only one bubble, which renders it uninformative for phasing, and we do not consider it further. The remaining phasing-informative reads in Figure 6.4 are represented as:

$$\mathcal{F} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{cc} l_1 & l_2 \end{array} \\ \begin{pmatrix} 0 & 0 \\ 2 & 2 \\ 1 & 2 \end{pmatrix} \end{array} \tag{6.1}$$

Corresponding to $\mathcal{F}$, we have a weight matrix $\mathcal{W} \in W^{|R|\times|L|\times m}$. Each entry in $\mathcal{W}(j, k)$ is a tuple storing a weight for each allele, which can for instance reflect "phred-scaled" (i.e. $-10\log(p)$) probabilities that the read supports a given allele. The weight of "0" at the i^{th} entry in the tuple $\mathcal{W}(j, k)$ encodes that the read r_j is aligned to allele path index i in bubble l_k. The remaining non-zero values in tuple $\mathcal{W}(j, k)$ store the confidence scores of switching the aligned read r_j to other alleles in bubble l_k.

For example, the corresponding weight matrix $\mathcal{W}(j, k)$ for $\mathcal{F}$ (6.1) is given by:

$$\mathcal{W} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{cc} l_1 & \quad l_2 \end{array} \\ \begin{pmatrix} [0, q_1, q_2] & [0, q_3, q_4] \\ [q_9, q_8, 0] & [q_{11}, q_5, 0] \\ [q_{10}, 0, q_7] & [q_5, q_6, 0] \end{pmatrix} \end{array} \tag{6.2}$$

where the entry $\mathcal{W}(1, 1)$ value $[0, q_1, q_2]$ means that the read r_0 is aligned to allele a_0 at bubble l_1. Additionally, the cost of flipping it to other alleles is q_1 for a_1 and q_2 for a_2.

We are now ready to present the problem formulation. The main insight is that solving phasing for bubble chains is similar to solving the phasing problem for multi-allelic SNVs in reference-based haplotype reconstruction. Therefore, we build on the previous formulation of the Minimum Error Correction (MEC) problem (Lancia et al., 2001) and its weighted version (wMEC) (Lippert et al., 2002; Patterson et al., 2015) and further adapt it to work on a subgraph consisting of a chain of bubbles, defining the *Minimum Error Correction for graphs* (gMEC) problem.

Problem 6.1 (wMEC for bubble chains (gMEC)). Assume we are given a bubble chain $L = (l_1, \ldots, l_{|L|})$ and a set R of aligned reads r_j that pass through these bubbles, with $\mathcal{F}(j, k)$ indicating the index of the allele in bubble l_k that the alignment of read r_j passes through, or "–" if it does not pass through l_k, and that $\mathcal{W}(j, k, i)$ is the cost of flipping $\mathcal{F}(j, k)$ to new value i. We want to find two paths through L, each of which consists of a sequence of allele indices specifying which allele the path takes in each bubble l_k, and then to flip entries of $\mathcal{F}$ such that each row is equal to one of the paths for all non-dash entries while the incurred costs are minimized.

Note that the wMEC problem constitutes a special case of gMEC, where the input graph is a chain of bi-allelic bubbles. Next, we describe how to solve gMEC via dynamic programming (DP), which is build upon the WhatsHap approach by Patterson et al. (2015) described in Section 5.4.

Solving gMEC for bubble chains. The basic idea is to now extend the dynamic program to consider all possible path-pairs through each bubble. In the bi-allelic case, we have only two paths in every bubble and, therefore, there is only one pair of distinct paths. In the multi-allelic case, we consider all possible path pairs in each bubble. The goal is to find an optimal pair of paths from the sequence graph

G_s. Analogously to the WhatsHap algorithm for wMEC, we proceed from left to right using dynamic programming.

To explain the dynamic programming algorithm that we use, consider a toy example with the weight matrix (6.2):

$$\mathcal{W} = \begin{array}{c} \\ r_1 \\ r_2 \\ r_3 \end{array} \begin{array}{c} \begin{array}{cc} l_1 & l_2 \end{array} \\ \begin{pmatrix} [0,10,5] & [0,5,8] \\ [7,6,0] & [5,2,0] \\ [2,0,4] & [4,3,0] \end{pmatrix} \end{array} \tag{6.3}$$

DP cell initialization. Along similar lines as Patterson et al. (2015), we first compute the local cost incurred by bipartition $B = (R, S)$ in column k, denoted $\Delta_C(k, B)$, and later combine it with the corresponding costs incurred in previous columns. The cost $W^i_{k,R}$ of flipping all entries in a read set R to an allele index $i \in \{0, 1, \ldots |l_k|\}$ is given by

$$W^i_{k,R} = \sum_{j \in R} [\![\mathcal{F}(j,k) \neq i]\!] \cdot \mathcal{W}(j,k,i),$$

In the same manner, we can compute costs $W^i_{k,S}$ for read set S to an allele index i.

To compute the cost incurred by a bipartition in a particular column k, we minimize over all possible pairs of alleles in bubble l_k. There are $\binom{|l_k|}{2}$ such pairs. So given the corresponding column vectors $\mathcal{F}(k)$ and $\mathcal{W}(k)$ of the bubble matrix and of the weight matrix, respectively, and the bipartition $B = (R, S)$ of active reads $A(k)$, the cost $\Delta_C(k, B)$ is computed by minimizing over all pairs of alleles $A = \{(x, y) \in l_k \times l_k | x \neq y, x < y\}$:

$$\Delta_C(k,B) = \min_{(p_0,p_1) \in A} \left\{ \min\{W^{p_0}_{k,S} + W^{p_1}_{k,R}, W^{p_1}_{k,S} + W^{p_0}_{k,R}\} \right\}, \tag{6.4}$$

where the outer minimization considers all allele pairs and the inner minimization considers the two possibilities of assigning those two alleles to the two haplotypes.

DP column initialization. We initialize the first DP column by setting $C(1, B) := \Delta_C(1, B)$ for all possible bipartitions B. We enumerate all bipartitions in Gray code order, as done previously in Patterson et al. (2015). This ensures that only one read is moved from one set to another in each step, facilitating constant time updates of the values $W^i_{k,S}$.

For a variant matrix (6.1) and its corresponding weight matrix (6.3), the DP column cell for bipartition $B = (R, S)$ is given by

$$\begin{aligned} \Delta_C(k,(R,S)) = \min \{ & W^0_{k,R} + W^1_{k,S}, W^1_{k,R} + W^2_{k,S}, \\ & W^0_{k,R} + W^2_{k,S}, W^1_{k,R} + W^0_{k,S}, \\ & W^2_{k,R} + W^1_{k,S}, W^2_{k,R} + W^0_{k,S} \} \end{aligned}$$

Now, plugging values from (6.3) into the above equation for different bipartitions, $\Delta_C(1, .)$ can be filled as follows:

$$\begin{aligned} &\Delta_C(1,(\{r_1,r_2,r_3\},\emptyset)) = \\ &\quad min\{9+0, 16+0, 9+0, 16+0, 9+0, 9+0\} = 9 \end{aligned}$$

Similarly, we can compute $\Delta_C(1, .)$ for other bipartitions $(\{r_1, r_2\}, \{r_3\})$, $(\{r_1, r_3\}, \{r_2\}), (\emptyset, \{r_1, r_2, r_3\}), (\{r_3\}, \{r_1, r_2\}), (\{r_2\}, \{r_1, r_3\})$.

Due to the use of the Gray code order, we can perform this operation for one DP column in $\mathcal{O}(\binom{|l_k|}{2} \cdot 2^{|A(k)|})$ time.

DP column recurrence. Note that $C(k, B)$ is the cost of an optimal solution of Problem 6.1 for input matrices restricted to the first k columns under the additional constraint that the solution's bipartition of the full read set extends B. Since column k lists all bipartitions, the optimal solution to the input matrix consisting of the first k columns would be given by the minimum in that column. To compute

Input : Set $A(1)$ of reads covering bubble l_1.
Output: $C(1, .)$
for *all bipartitions B of column k* **do**
  Compute $\Delta_C(k, B)$ using Equation 6.4 and store in $C(1, B)$.

Algorithm 2: DP COLUMN INITIALIZATION

entries in column $C(k+1, \cdot)$, we add up local costs incurred in column $k+1$ and costs from the previous column (see Algorithm 3). To adhere to the semantics of $C(k+1, B)$ described above, only entries in column k whose bipartitions are *compatible* with B are to be considered as possible "predecessors" of $C(k+1, B)$.

Definition 6.6 (Bipartition compatibility). For bipartitions $B = (P, Q)$ of A and $B' = (P', Q')$ of A', B and B' are *compatible* if $P \cap A \cap A' = P' \cap A \cap A'$ and $Q \cap A \cap A' = Q' \cap A \cap A'$, denoted by $B \simeq B'$

For example, consider the second column from (6.1) and (6.3). Let us compute $C(2, .)$ for different bipartitions using recurrence in Algorithm 3:

$$C(2, (\{r_1, r_2, r_3\}, \emptyset)) = \min\{9+0, 10+0, 9+0, 10+0, 8+0, 8+0\}$$

$$+ \min\{C(1, (\{r_1, r_2, r_3\}, \emptyset)\} = 8 + 9 = 17$$

To fill DP column $C(2, .)$, we can analogously compute this for the remaining bipartitions $(\{r_1, r_2\}, \{r_3\})$, $(\{r_1, r_3\}, \{r_2\})$, $(\emptyset, \{r_1, r_2, r_3\})$, $(\{r_3\}, \{r_1, r_2\})$, and $(\{r_2\}, \{r_1, r_3\})$.

Input : $C(1, .)$ for all bipartitions of bubble k.
Output: $C(k, .)$ for all the columns k up to the last column $|L|$
for *all columns $k \in \{2 \ldots |L|\}$* **do**
  for *all bipartitions $B \in \mathcal{B}(A(k))$* **do**
    Compute $\Delta_C(k, B)$ using Equation 6.4.
    Combine it with cost from column $k-1$ to obtain cost for column k:

$$C(k, B) = \Delta_C(k, B) + \min_{B' \in \mathcal{B}(A(k-1)) : B \simeq B'} C(k-1, B')$$

  where $\mathcal{B}(A(k))$ denotes the set of all bipartitions of $A(k)$ and $B \simeq B'$ follows Definition 4.1.

Algorithm 3: DP TABLE

Backtracing. We can backtrace from the last column $C(|L|, \cdot)$ to compute an optimal bipartition $B = (R, S)$ of all input reads. Given this bipartition, we obtain minimum-cost haplotypes as follows: Let $B_k = (R_k, S_k)$ with $R_k = R \cap A(k)$ and $S_k = S \cap A(k)$ be the induced bipartition in column k. We then set

$$h_0(k) = a_i \quad \text{with } i := \operatorname*{arg\,min}_{i' \in \{0,1,\ldots|l_k|\}} W^{i'}_{k,R_k},$$

$$h_1(k) = a_j \quad \text{with } j := \operatorname*{arg\,min}_{j' \in \{0,1,\ldots|l_k|\}} W^{j'}_{k,S_k},$$

where a_i and a_j refer to the corresponding allele paths of bubble k (see Definition 6.2).

Time complexity. Computing one DP column takes $\mathcal{O}(\binom{m}{2} \cdot 2^{|A(k)|})$ time, and the total running time is $\mathcal{O}(\binom{m}{2} \cdot 2^{|A(k)|} \cdot |L|)$ for $|L|$ bubbles, where m is the maximum number of alleles in any bubble from L. Running time is independent of read-length and, therefore, the algorithm is suitable for the increased read lengths available from upcoming sequencing technologies.

6.3.6 Generation of final assemblies

To generate final assemblies, for every connected component in the base sequence graph G_s, we traverse along the haplotype paths (h_0, h_1) running through that component. For the nodes in each path, we concatenate together the nodes' sequences from the base sequence graph G_s (in either in their forward or reverse-complement orientations, as specified by the path) in order to generate the final haplotig sequences.

6.4 Datasets and experimental setup

To evaluate the performance of our method, we consider the real data available from two haploid yeast strains SK1 and Y12 (Yue et al., 2017), which we combine to generate a pseudo-diploid yeast. Both the SK1 and Y12 yeast strains are deeply sequenced using Illumina and PacBio sequencing. The Illumina dataset is sequenced to an average coverage of 469× with 151 bp paired end reads. We randomly downsample the dataset to a lower average coverage of 50×. The PacBio data is sequenced to an average coverage of 334× with an average read length of 4510 bp. For coverage analysis, we randomly downsample the PacBio reads to obtain datasets of different coverages 10×, 20× and 30× with their average read-lengths of 4482, 4501 and 4516 bp respectively.

6.4.1 Pipeline implementation

Sequence graph. The first step in our pipeline is to perform error correction on the Illumina data by using BFC (Li, 2015a), which, in our experience, retains heterozygosities well for diploid genomes. BFC is used with default parameters and provided with a genome size of 12.16 Mbp. The second step is to generate a sequence graph that includes heterozygosity information. To construct such a graph, we first construct the assembly graph by using a modified version of SPAdes v3.10.1 (Bankevich et al., 2012). We modify the original SPAdes to skip the bubble removal step and retain the heterozygosity information in the graph, and run it with default parameters plus the `--only-assembler` option. It uses the short Illumina reads to generate a De Bruijn-based assembly graph without any error correction. We then convert the assembly graph to a bluntified sequence graph using VG (Garrison et al., 2017). After graph simplification, the resulting sequence graph has 158,567 nodes and 190,767 edges.

Bubble detection. In the next stage, we use VG's snarl decomposition algorithm (Paten et al., 2017) to detect the regions of heterozygosity, or *snarls*, in the sequence graph. This results in 29,071 bubbles.

PacBio Alignments. After bubble detection, we align different coverage levels (10×, 20× and 30×) of long read PacBio data to the generated sequence graph using GraphAligner[2]. This resulted in 21,868, 43,459 and 73,129 PacBio alignments for input coverages of 10×, 20× and 30×, respectively.

Bubble ordering. To obtain an ordering of bubbles, we perform *de novo* assembly using Canu v1.5 (Koren et al., 2017) on each PacBio dataset. As suggested by Giordano et al. (2017), we use Canu v1.5 with the following parameter values: `corMhapSensitivity=high, corMinCoverage=2, correctedErrorRate=0.10, minOverlapLength=499, corMaxEvidenceErate=0.3`. Next, we align these Canu contigs to the sequence graph to obtain the bubble ordering, which we define as the sequence of bubbles encountered by each aligned contig. Note that we use Canu solely for bubble ordering. In this paper, we restrict ourselves to phasing bubbles only in unique, non-repetitive regions. We detect repetitive bubbles based on the coverage depth of the PacBio alignments and remove them from downstream analyses. The coverage depth threshold used is 1.67 times the average coverage. This results in 148, 80, and 71 bubble chains, and 26,576, 27,556 and 27,741 bubbles, at coverages of 10×, 20×, and 30× respectively.

[2] https://github.com/maickrau/GraphAligner

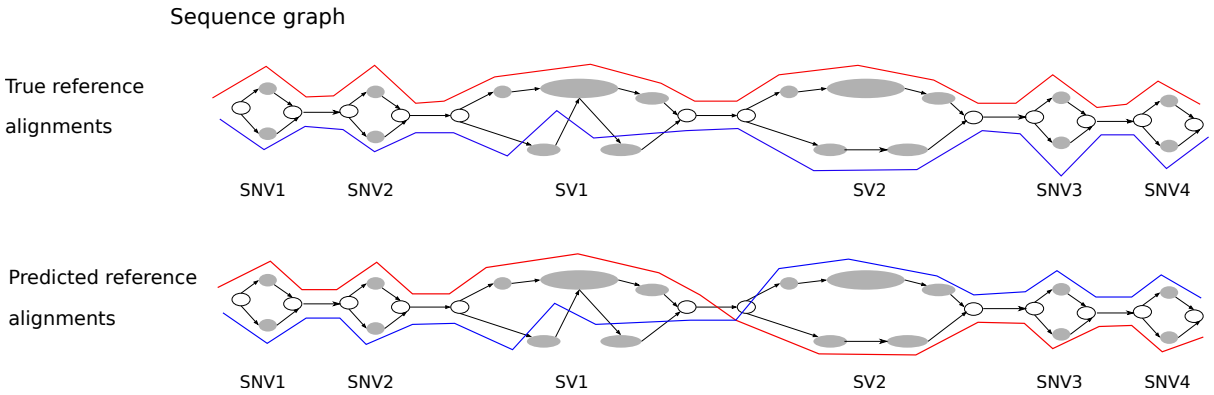

Figure 6.5: For a subgraph of G_s, this example shows the true (top) and predicted (bottom) versions of two haplotype alignments (red and blue) through a series of bubbles. When comparing the correspondingly-colored lines between the two versions, we see one switch between SV1 and SV2: the prediction contains one switch error. Six bubbles have been phased, for a total of five phase connections between consecutive bubbles. Therefore, the phasing error rate is 1/5.

Graph-based phasing. For each of the coverage conditions, we take as input the ordered bubbles, the long-read PacBio alignments and the sequence graph, and solve the gMEC problem by assuming constant weights in the weight matrix $\mathcal{W}$. The optimal bipartition is computed via backtracing and the final haplotigs are generated by concatenating the node labels of the two optimal paths. These steps have been implemented in our WhatsHap software as a subcommand `phasegraph`[3].

6.4.2 Running Falcon Unzip

The main goal of this study is to measure the performance of phasing using a graph based approach, and, in particular, the quality of haplotypes at heterozygous sites achievable by using this method with low coverage PacBio data. Therefore, we compared our graph-based approach to the state-of-the-art contig based phasing method Falcon Unzip, which also generates diploid assemblies.

The Falcon Unzip (Chin et al., 2016) algorithm first constructs a string graph composed of "haploid consensus" contigs, with bubbles representing structural variant sites between homologous loci. Sequenced reads are then phased and separated for each haplotype on the basis of heterozygous positions. Phased reads are finally used to assemble the backbone sequence (primary contigs) and the alternative haplotype sequences (haplotigs). The combination of primary contigs and haplotigs constitutes the final diploid assembly, which includes phasing information dividing single-nucleotide polymorphisms and structural variants between the two haplotypes.

We ran Falcon Unzip using the parameters given in the official parameter guide[4]. We tried to run Falcon Unzip for lower coverages of 10× and 20×, but it did not generate output in these cases (and we assume it is not designed for such low coverages). Therefore, we only ran Falcon Unzip for 20× PacBio coverage. Primary contigs and haplotigs were polished using the Quiver algorithm and corrected for SNPs and indels using Illumina data via Pilon, with the parameters "`--diploid`" and "`--fix all`" (Walker et al., 2014).

6.4.3 Assembly performance assessment

To evaluate the accuracy of the predicted haplotypes, we align reference assemblies of the two yeast strains SK1 and Y12 (Yue et al., 2017) to the sequence graph. We emphasize that these reference

[3]Presently this functionality resides in the `MAV` branch, which will be merged to the `master` branch in the near future and will be part of future WhatsHap releases.

[4]http://pb-falcon.readthedocs.io/en/latest/parameters.html

assemblies are only used for evaluation purposes and are not a part of our assembly pipeline. We use the following performance measures for the evaluation of diploid assemblies:

Phasing error rate. Over the yeast genome, we compare the different diploid assemblies with the ground truth haploid genomes of SK1 and Y12. As with the reference assemblies, we align the haplotigs produced by Falcon Unzip to our sequence graph. For each phased bubble chain, the predicted haplotype is expressed as a mosaic of the two true haplotypes, minimizing the number of switches. This minimum then gives the number of switch errors. The phasing error rate is defined as the number of switch errors divided by the number of phased bubbles. Figure 6.5 illustrates this calculation for a toy example. The top panel shows the true references aligned to the sequence graph. At the bottom, predicted haplotypes (from Falcon Unzip or our graph-based approach) are aligned to the graph. Comparing the true and predicted haplotypes, we see one switch between SV1 and SV2, which means that the switch error count is one. The number of phase connections between consecutive bubbles is five and the resulting switch error rate for this example is 1/5.

Average Percent Identity. We consider the best assignment of each haplotig to either of the two true references, obtained by aligning the haplotig to the references. For each whole diploid assembly, we compute the average of the best-alignment percent identities over all haplotigs.

Assembly contiguity. We assess the contiguity of the assemblies by computing the N50 of haplotig size.

Assembly completeness. We consider two assembly completeness statistics: first, the total length of haplotigs assembled by each method, and second, the total number of unphased contigs.

6.5 Results

In this section, we present the results of our analysis of the diploid assemblies generated by our method and by Falcon Unzip on the data sets described above.

Coverage analysis. To discover a cost-effective method for assembling a diploid genome, we consider PacBio datasets that vary in terms of coverage—specifically, 10×, 20× and 30× coverage are considered. One of the primary aims of our study is to compare two approaches—the graph-based approach we implemented and the contig-based phasing done by Falcon Unzip. In doing so, we quantify the agreement between the diploid assemblies generated by both methods and the true references. Table 6.1 shows the assembly performance statistics for both of these methods. In order to assess the accuracy of the competing diploid assemblies, we compute the phasing error rate and the average percent identity at different PacBio coverages. For the graph-based approach, we observe that as we increase the long read coverage from 10× to 30×, the average identity of haplotigs increases from 99.5% to 99.8% and the phasing error rate decreases from 2.5% to 0.7%. In contrast, Falcon Unzip produces haplotigs with an average identity of 99.4% and phasing error rate of 3.8% at 30× coverage. Overall, comparing the agreement between the graph-based approach (at 10× coverage) and Falcon Unzip (at 30× coverage) to the true references, our graph-based approach delivers better haplotigs with respect to all measures reported in Table 6.1. We believe that one reason for this is that we use an Illumina-based graph as a backbone. Furthermore, optimally solving the gMEC formulation of the phasing problem most likely contributes to generating accurate haplotigs. Overall, our analysis supports the conclusion that our approach delivers accurate haplotype sequences even at a long read coverage as low as 10×.

To analyse the effect of different coverages of the Illumina short-read datasets on the quality of our haplotigs, we went back to the original, high coverage Illumina dataset (which we had been using downsampled to 50× coverage) and downsampled it to 100× coverage, i.e. twice the amount of reads used above. We observed that increasing the coverage did not have a drastic effect on the quality of haplotigs. The average phasing identity rose to 99.81% and the total haplotig size was 23.9 Mbp, which is virtually identical to the results for 50× as reported in Table 6.1.

With an increase in average PacBio coverage from 10× to 30×, the haplotype contiguity achievable by using our approach improves from 40 kbp to 43 kbp. By way of comparison, Falcon Unzip delivers haplotigs with a N50 length of 32 kbp at the same coverage level. This highlights the fact that our

Statistics	PacBio coverage	Graph-based approach	Falcon Unzip
Diploid assemblies Quality			
Average Identity[%]	10×	99.50	—
	20×	99.61	—
	30×	99.80	99.4
Phasing error rate[%]	10×	2.5	—
	20×	1.5	—
	30×	0.7	3.8
Contiguity			
N50 haplotig size [bp]	10×	40k	—
	20×	42k	—
	30×	43k	32k
Completeness			
Haplotig size [Mbp]	10×	20.7	—
	20×	21.1	—
	30×	23.9	16.6
# Unphased contigs	10×	2	—
	20×	2	—
	30×	2	77

Table 6.1: Comparison of two phasing methods, Falcon Unzip and our graph-based approach, at different PacBio coverage levels. For computing the "haplotig N50", we only consider those portions of a contig for which two haplotypes are available, i.e. those regions where Falcon reports both a primary contig and an alternative haplotig. For "haplotig size", we sum the length of contigs on both haplotypes ("primary contigs" plus "haplotigs" in terms of Falcon's output), so the target size is twice the genome size (24.3Mbp in case of yeast).

approach generates more contiguous haplotypes compared to Falcon Unzip. In terms of haplotype completeness, our approach yields diploid assemblies of length 20.7 Mbp, 21.1 Mbp and 23.9 Mbp at average PacBio coverages of 10×, 20× and 30× respectively. At coverage 30×, Falcon Unzip delivers a total assembly size of 16.6 Mbp, while the total length of both haplotypes of the pseudo-diploid yeast genome is 24.3 Mbp. Our approach therefore delivers more complete haplotypes at a long-read coverage of 10× compared to Falcon Unzip at a coverage of 30×. There are 2 haplotigs that are not phased by our approach; this is due to the lack of heterozygosity over those regions. In comparison, there are 77 (out of 123) contigs that are not phased by Falcon Unzip. In summary, our graph-based approach delivers complete and contiguous haplotype sequences even at a relatively low coverage of 10×.

Bubble characterization. We attempted to characterize the nature of the heterozygous genomic variation encoded in the phased bubbles. There are 25,033 bi-allelic bubbles phased by our approach when using 30× coverage PacBio data. Of these bubbles, there are 15,293 for which both allele sequences have a length of at most 1 bp, out of which 15,258 are single base pair substitutions (SNVs) and 35 are 1 bp indels. The remaining 9,740 bubbles either encode two or more small variants or more complex differences. To differentiate these cases, we computed an alignment between the two allele paths and refer to those bubbles for which the alignment contains only substitutions but no indels as "pure substitutions". Figure 6.6a shows the joint distribution of length and (Hamming) distance for these pure substitution bubbles. This analysis reveals, on the one hand, that many longer pure substitutions have a low distance and hence encode multiple SNVs and, on the other hand, that there also exists a population of more complex substitutions. For the 1,489 bubbles not classified as pure substitutions, which we refer to as "mixed bubbles", Figure 6.6b shows the absolute length difference between the two

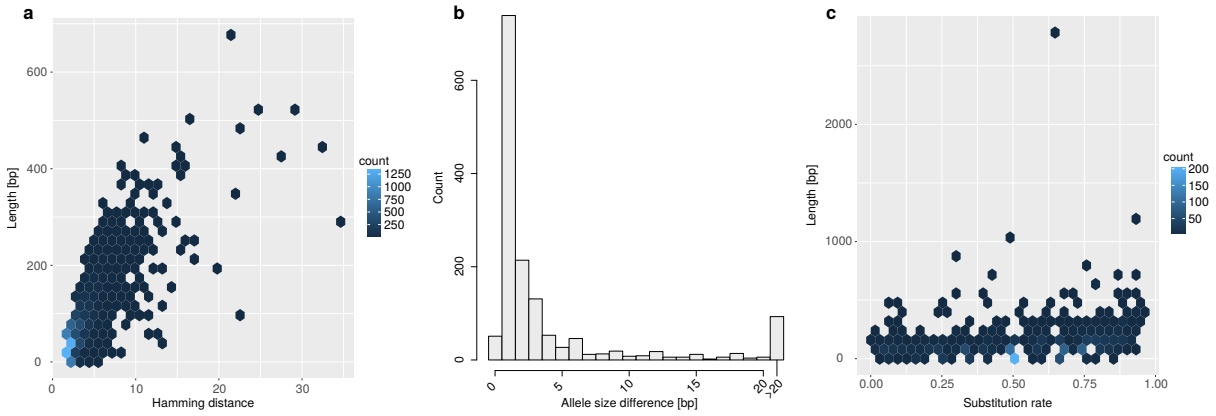

Figure 6.6: Structural variation analysis of phased bubbles from our graph-based approach. a: Joint distribution of allele length and Hamming distance, for pure substitutions. b. Distribution of size difference between the two alleles, for mixed bubbles and indels. Pure substitutions always have a size difference of 0, and are not included in the figure. c. Joint distribution of the length of the longer allele and the substitution rate, for mixed bubbles. With a higher substitution rate, the bubble has more substitutions, and with a lower rate more indels.

alleles. While this difference is small for most bubbles, there are 93 bubbles with a length difference of 21 bp or more. To further elucidate the nature of the sequence differences, Figure 6.6c presents the joint distribution of length of the longer allele and substitution rate, which is defined as the fraction of substitutions among all edit operations done to align the two sequences. (That is, a pure insertion or deletion has a substitution rate of 0.)

6.6 Discussion

The Falcon Unzip method (Chin et al., 2016) is based purely on PacBio reads, which exhibit a high error rate; it is therefore not suitable for lower coverages. By using (costly) high coverage PacBio data, Falcon Unzip can generate good quality assemblies with an average haplotig identity of up to 99.99% (Chin et al., 2016). However, it follows a conservative approach for phasing genomic variants. As sketched in Figure 6.1, Falcon Unzip generates long primary contigs, but tends to phase them only partially.

To address the above problems, we have created a novel graph-based approach to diploid genome assembly that combines different sequencing technologies. By using one technology producing shorter, more accurate reads, and a second technology delivering long reads, we produce accurate, complete and contiguous haplotypes. Our method also provides a cost-effective way of generating high quality diploid assemblies. By performing phasing directly in the space of sequence graphs—without flattening them into contigs in intermediate steps—we can phase large structural variants, which is not possible using linear approaches. We have tested our approach using real data, in the form of a pseudo-diploid yeast genome, and we have shown that we deliver accurate and complete haplotigs. Furthermore, we have shown that we can detect and phase structural variants.

In this study, we restricted ourselves to phasing unique, non-repetitive regions of the genome. As a next step, we plan to develop techniques for phasing repetitive regions as well. Resolving repeats and polyploids phasing are closely related problems, as pointed out by Chaisson et al. (2017a). Therefore, we will aim to solve heterozygous variants and repeats in a joint phasing framework, in order to obtain even more contiguous diploid genome assemblies that include both types of features. The machinery we plan to develop for this purpose would also remove the need to run an external assembler (Canu)

for bubble ordering. Finally, our framework allows, in principle, for incorporating additional data from other sequencing technologies, such as chromatin conformation capture (Burton et al., 2013), linked read sequencing (Weisenfeld et al., 2017), and single-cell template strand sequencing (Strand-seq; Porubský et al., 2016). Chapter 4 on reference-based haplotyping, we have shown such integrative approaches to be very powerful when inferring chromosome-scale haplotypes; we believe similar results can be obtained for *de novo* diploid genome assemblies.

This chapter will appear at ISMB Proceedings/Bioinformatics 2018, which has co-authors as Mikko Rautiainen, Adam M Novak, Erik Garrison, Richard Durbin & Tobias Marschall. The figures in this chapter are taken from the paper. My contribution involves in developing the phasing method and writing the paper.

Chapter 7

Contributions and discussion

We summarize our algorithmic contributions for addressing the four open problems introduced in Chapter 1. Furthermore, we provide broader perspectives on how these approaches can be utilized to answer different biological questions and further help in future precision medicine.

7.1 Contributions

Deriving accurate and complete haplotypes from sequencing datasets helps in genome-engineering, evolutionary studies, and functional and comparative genomics. In this thesis, we have provided efficient algorithms to perform haplotyping based on different information sources.

There are two ways to perform haplotyping using sequencing data: reference-based phasing and haplotype-aware de novo assembly. In reference-based phasing, reads are aligned to the reference genome and then the phasing is performed using aligned reads over the variants. Reference-based phasing for a single individual is formulated as the Minimum Error Correction (MEC) and its weighted version wMEC (as explained in Chapter 1). Some sequencing technologies give rise to Gapless-MEC instances (e.g. PacBio, ONT), while other technologies produce general MEC instances (e.g. Strand-seq, 10X genomics). To study the complexity and approximation status of these instances is very important, and helps in unraveling and understanding the structures.

7.1.1 Approximation status of Gapless-MEC

Third-generation sequencing technologies generate single-ended reads in the order of 10Kb in length. These reads are often aligned to the reference genome for performing phasing. The alignment of these reads can be mathematically represented in the form of a matrix, which is denoted by Gapless-MEC. Gapless-MEC is a generalization for Binary-MEC because Binary-MEC instances contain only binary values, whereas Gapless-MEC instances are allowed to contain wildcards at the end of each row. It is known that Binary-MEC is in PTAS and MEC is APX-hard. However, there is a gap in our knowledge of the approximation status of Gapless-MEC, which has been open for 10 years. As a part of this thesis, we study the approximation status of Gapless-MEC and conclude that Gapless-MEC is in QPTAS. Proving Gapless-MEC to be in QPTAS implies that the it is not APX-hard.

The main challenge in deriving a QPTAS for Gapless-MEC consists in sampling sufficiently many rows such that we get haplotypes from end-to-end, otherwise we might miss parts which may lead to an unbounded approximation. To handle this problem, we provided a dynamic programming approach that captures the structure of the considered instances.

We started with simple instances (SWC) such that all the rows start from one column and are ordered by increasing length of the binary part. Such instances are simple to solve because we can apply the Binary-MEC algorithm for the sub-matrices starting from column one and then argue that we generate good solution strings. As soon as we are at a sub-instance consisting of wildcards and binary values, we carefully sample more rows and consider a weighted majority to generate haplotypes.

Furthermore, we show that in expectation, the algorithm works well at each column and is in PTAS. We have generalized this algorithm to solve complete SWC instances by using clever DP algorithms.

We further generalized this DP algorithm for SWC instances to solve sub-interval free instances. The main insights are that we can determine a sequence of columns. At each column, we get SWC instances at its left and right side. In order to combine the consecutive columns, we introduced the notion of dominance such that we can solve two consecutive columns simultaneously in the DP, still getting a good approximation. We proved that this dynamic programming algorithm is in PTAS.

Next, we generalize this dynamic programming algorithm to solve general Gapless-MEC instances. The main observation is that we can divide the instances into different length classes and each length class can be solved in quasi-polynomial time. To combine different length class instances, we use clever techniques in DP and prove that the generalized dynamic program is in QPTAS.

7.1.2 Parameterized algorithm for phasing individual genomes

There are different sequencing technologies available to sequence the genome, one of them is single-cell template strand sequencing (Strand-seq) and others are long-read technologies. The strand-specific technology provides global but sparse information about the haplotypes. The main advantage of Strand-Seq technology is that it provides Illumina reads along with directionality information. The downside of Strand-Seq data is its sparseness, which creates a necessity it to integrate with other available sources such as PacBio and ONT to get complete haplotypes.

We have presented an integrative phasing strategy based on the parameterized WhatsHap algorithm to generate accurate, contiguous and complete haplotypes. The parameterized algorithm has coverage as a parameter, which is usually small enough in practice that makes it feasible to solve these instances in short time. We have provided the integrative framework that consists of using MEC for data integration. We have used a dynamic programming by Patterson et al. (2015) to solve these instances. The main idea is to go column wise from left to right. At each column, we store the best allele assignment score for each bipartition. In the recurrence step, when we compute the bipartition at $k + 1$ column, we consider the bipartition from the previous column k such that the reads assigned to different haplotypes remain the same over two consecutive columns and we store the best allele assignment for the corresponding bipartition in DP table. In this way, we recurse until the last column and finally backtrace to generate the haplotypes.

We have demonstrated the effectiveness of this algorithm on the real human genome. We have performed a comprehensive analysis on the level of coverage required from each sequencing technology in this integrative framework.

In the situation, when the Strand-Seq data is not available to provide global information about the haplotypes, it is better to utilize the information from pedigrees.

7.1.3 Parameterized algorithm for phasing pedigrees

For a pedigree of genomes, we have extended the parameterized algorithm for a single individual to additionally make use of the pedigree information. In this model, we want to find two haplotypes for each individual in a pedigree such that we obey the Mendelian laws of inheritance. We therefore formulated the combination of read-based phasing and genetic haplotyping into an integrative framework that we call MEC on pedigrees (PedMec).

As input, we have SNP matrices and corresponding weight matrices for each individual in a pedigree. Additionally, we have the recombination vector that represents the likelihood of recombination between every pair of consecutive variants. We have provided a dynamic programming algorithm to solve PedMec instances. The major difference between the pedigree framework and the single individual case consists in the number of partitions. In our pedigree framework, we have four partitions, instead of two, because the reads from the mother, father and child each can be partitioned into two sets, with additional constraints that each read from the child partitions corresponds to either mother or father. The constraints are represented by the transmission value, which determines which haplotype from

each parent is transmitted to the child. We compute the best allele assignment score for these four partitions at each column and store them in a DP table. The recurrence step proceeds similarly to the single individual case, but with additional step of trying all possibilities of recombination in mother or father. We continue this process until the last column and backtrace to find haplotypes for each individual in a pedigree.

The running time of pedigree algorithm is linear in 2^t, where t is the number of trio-relationships, but it still remains linear in the number of variants.

We demonstrate the effectiveness of our algorithm on both real and simulated data. We considered the AJ trio from Genome in a Bottle Consortium (GIAB). We showed that we provided long-range chromosomal-length accurate haplotypes by incorporating trio information. The main conclusion is we require low coverages of $2\times$ for each individual with family relationship information as opposed to a single individual at $15\times$ coverage.

7.1.4 Haplotype-aware de novo assembly

In all the approaches for reference-based haplotyping, we performed phasing based on the reads aligned to the reference genome. Therefore, there is a reference bias. We have proposed a new approach to reconstructing the genome sequences of diploid organisms directly from the reads, without relying on a reference genome. The process of obtaining haplotype sequences from reads is called haplotype-aware denovo assembly. Constructing these sequences is very important to understanding the true characteristics of diploid organisms.

Current assemblers collapse the heterozygosity information represented by bubbles in assembly graphs and therefore generate a haploid consensus sequence. A popular diploid assembler — Falcon Unzip, which is purely PacBio based — has the ability to generate diploid assemblies. The main disadvantages of Falcon Unzip method are that it fails for low coverage datasets and for genomes with a high heterozygosity rate.

In order to handle these problems, we have provided a novel graph-based diploid assembly pipeline, which makes use of short accurate and long error-prone sequencing reads. Our pipeline involves constructing an assembly graph using Illumina data such that it acts as a backbone for subsequent steps. This allows us to accurately detect SNVs represented by bubbles in the assembly graph. Furthermore, we aligned the long reads to this graph to span over the bubbles and detect bubble chains. By using these alignments of long reads to bubble chains, we perform phasing using a generalized wMEC model, which is based on the observation that phasing bubble chains is equivalent to phasing multi-allelic variants. Compared to solving wMEC with the WhatsHap algorithm, the running time increases by a factor of $\binom{a}{2}$, where a is the maximum number of alleles in any bubble.

We have demonstrated the effectiveness of our algorithm by combining two yeast strains to form a pseudo-diploid yeast genome. We have shown that our graph-based approach has the ability to generate more accurate, contiguous and complete assemblies even at a low coverage of $10\times$. Additionally, we have pointed out that we can detect and phase large structural variations.

7.2 Discussion

Third generation sequencing technologies have vastly increased our ability to address the haplotyping problem. Using long reads delivered by these technologies, the algorithms we designed are able to generate accurate, contiguous and complete haplotypes. There remain some complex regions in diploid genomes that are not yet solved by current approaches, either reference-based or *de novo*. We believe that both reference-based and de novo approaches can be further improved to solve substantially more complex and repetitive regions. In a *de novo* context, different graph parameters such as tree-width, maximum branching factor, copy number or other parameters can be explored to solve the complex regions of genome. In a reference-based context, ILP, a greedy heuristic or other parameterized approaches have the power to solve complex instances efficiently.

The techniques developed in this thesis can potentially also help to solve genomes of higher ploidy. For such genomes, the current WhatsHap algorithm leads to running time of k^c, where k is the ploidy and c is the coverage. More efficient ways could be explored to further improve the running time by pruning the search space. Another conceivable extension of the WhatsHap algorithm is to separate species based on long read data, thus solving the popular meta-genome assembly problem. Beyond that, the techniques we have developed could have utility for transcriptome assembly of RNA sequences. In all these problem, the main challenge is scalability, and therefore, efficient data structure and algorithms need to be developed to address these challenges.

Phasing large structural variants in complex regions during assembly was not possible earlier. These variants include larger genomic ranges by substitution, insertion, deletion, copy number. Haplotype-aware approaches from long read data can help in detecting structural variants in complex regions and perform complete SV analyses.

In a general view, it is expected that in coming years reads become even longer and error rates will be reduced. Constant algorithmic efforts are required to handle the datasets that will arise from new sequencing technologies. The methods proposed in this thesis could be refined, new applications pursued, and other data types could be integrated.

By making algorithmic contributions to haplotyping, this thesis provides the genomics community with computational tools to attack important biological questions such as haplotype-resolved studies of genetic variation and genome instability. Accurate and complete haplotypes help in investigating mechanisms behind complex phenotypes in humans, including ageing and common diseases such as cancer. Having access to haplotype information promises insights into biological mechanisms of disease, which further translates into potential advancements of precision medicine. Also, haplotype knowledge is useful in population genetics, in particular for answering question of how evolution has shaped genomic architecture.

In summary, haplotype information is essential for the complete description of individual genomes. Access to haplotype resolved genomes enables us to discover novel genome features that were previously hidden. The computational tools developed in this thesis enable haplotype-aware analyses and open up new perspectives to answer open biological questions and to deepen our understanding of functioning of genomes.

Bibliography

Abecasis, G. R., Cherny, S. S., Cookson, W. O., and Cardon, L. R. (2002). Merlin—rapid analysis of dense genetic maps using sparse gene flow trees. *Nature genetics*, 30(1):97.

Aguiar, D. and Istrail, S. (2012). Hapcompass: a fast cycle basis algorithm for accurate haplotype assembly of sequence data. *Journal of Computational Biology*, 19(6):577–590.

Aguiar, D. and Istrail, S. (2013). Haplotype assembly in polyploid genomes and identical by descent shared tracts. *Bioinformatics*, 29(13):i352–i360.

Alon, N. and Sudakov, B. (1999). On two segmentation problems. *Journal of Algorithms*, 33(1):173–184.

Ammar, R., Paton, T. A., Torti, D., Shlien, A., and Bader, G. D. (2015). Long read nanopore sequencing for detection of hla and cyp2d6 variants and haplotypes. *F1000Research*, 4.

Antipov, D., Korobeynikov, A., McLean, J. S., and Pevzner, P. A. (2015). hybridspades: an algorithm for hybrid assembly of short and long reads. *Bioinformatics*, 32(7):1009–1015.

Bankevich, A., Nurk, S., Antipov, D., Gurevich, A. A., Dvorkin, M., Kulikov, A. S., Lesin, V. M., Nikolenko, S. I., Pham, S., Prjibelski, A. D., et al. (2012). Spades: a new genome assembly algorithm and its applications to single-cell sequencing. *Journal of computational biology*, 19(5):455–477.

Bansal, V. and Bafna, V. (2008). HapCUT: an efficient and accurate algorithm for the haplotype assembly problem. *Bioinformatics*, 24(16):i153–i159.

Bansal, V., Halpern, A. L., Axelrod, N., and Bafna, V. (2008). An mcmc algorithm for haplotype assembly from whole-genome sequence data. *Genome research*, 18(8):1336–1346.

Bashir, A., Klammer, A. A., Robins, W. P., Chin, C.-S., Webster, D., Paxinos, E., Hsu, D., Ashby, M., Wang, S., Peluso, P., et al. (2012). A hybrid approach for the automated finishing of bacterial genomes. *Nature biotechnology*, 30(7):701–707.

Ben-Bassat, I. and Chor, B. (2014). String graph construction using incremental hashing. *Bioinformatics*, 30(24):3515–3523.

Ben-Elazar, S., Chor, B., and Yakhini, Z. (2016). Extending partial haplotypes to full genome haplotypes using chromosome conformation capture data. *Bioinformatics*, 32(17):i559–i566.

Bentley, D. R., Balasubramanian, S., Swerdlow, H. P., Smith, G. P., Milton, J., Brown, C. G., Hall, K. P., Evers, D. J., Barnes, C. L., Bignell, H. R., et al. (2008). Accurate whole human genome sequencing using reversible terminator chemistry. *nature*, 456(7218):53–59.

Berlin, K., Koren, S., Chin, C.-S., Drake, J. P., Landolin, J. M., and Phillippy, A. M. (2015). Assembling large genomes with single-molecule sequencing and locality-sensitive hashing. *Nature biotechnology*, 33(6):623.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426.

Bonizzoni, P., Dondi, R., Klau, G. W., Pirola, Y., Pisanti, N., and Zaccaria, S. (2016). On the minimum error correction problem for haplotype assembly in diploid and polyploid genomes. *Journal of Computational Biology*.

Bowe, A., Onodera, T., Sadakane, K., and Shibuya, T. (2012). Succinct de bruijn graphs. In *International Workshop on Algorithms in Bioinformatics*, pages 225–235. Springer.

Browning, S. R. and Browning, B. L. (2007). Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. *The American Journal of Human Genetics*, 81(5):1084–1097.

Browning, S. R. and Browning, B. L. (2011). Haplotype phasing: existing methods and new developments. *Nature Reviews Genetics*, 12(10):703–714.

Burton, J. N., Adey, A., Patwardhan, R. P., Qiu, R., Kitzman, J. O., and Shendure, J. (2013). Chromosome-scale scaffolding of de novo genome assemblies based on chromatin interactions. *Nature biotechnology*, 31(12):1119.

Butler, J., MacCallum, I., Kleber, M., Shlyakhter, I. A., Belmonte, M. K., Lander, E. S., Nusbaum, C., and Jaffe, D. B. (2008). Allpaths: de novo assembly of whole-genome shotgun microreads. *Genome research*, 18(5):810–820.

Casillas, S. and Barbadilla, A. (2017). Molecular population genetics. *Genetics*, 205(3):1003–1035.

Chaisson, M. J., Mukherjee, S., Kannan, S., and Eichler, E. E. (2017a). Resolving multicopy duplications de novo using polyploid phasing. In *International Conference on Research in Computational Molecular Biology*, pages 117–133. Springer.

Chaisson, M. J., Wilson, R. K., and Eichler, E. E. (2015). Genetic variation and the de novo assembly of human genomes. *Nature reviews. Genetics*, 16(11):627.

Chaisson, M. J. P., Sanders, A. D., Zhao, X., Malhotra, A., Porubsky, D., Rausch, T., Gardner, E. J., Rodriguez, O., Guo, L., Collins, R. L., Fan, X., Wen, J., Handsaker, R. E., Fairley, S., Kronenberg, Z. N., Kong, X., Hormozdiari, F., Lee, D., Wenger, A. M., Hastie, A., Antaki, D., Audano, P., Brand, H., Cantsilieris, S., Cao, H., Cerveira, E., Chen, C., Chen, X., Chin, C.-S., Chong, Z., Chuang, N. T., Church, D. M., Clarke, L., Farrell, A., Flores, J., Galeev, T., David, G., Gujral, M., Guryev, V., Haynes-Heaton, W., Korlach, J., Kumar, S., Kwon, J. Y., Lee, J. E., Lee, J., Lee, W.-P., Lee, S. P., Marks, P., Valud-Martinez, K., Meiers, S., Munson, K. M., Navarro, F., Nelson, B. J., Nodzak, C., Noor, A., Kyriazopoulou-Panagiotopoulou, S., Pang, A., Qiu, Y., Rosanio, G., Ryan, M., Stutz, A., Spierings, D. C. J., Ward, A., Welsch, A. E., Xiao, M., Xu, W., Zhang, C., Zhu, Q., Zheng-Bradley, X., Jun, G., Ding, L., Koh, C. . L., Ren, B., Flicek, P., Chen, K., Gerstein, M. B., Kwok, P.-Y., Lansdorp, P. M., Marth, G., Sebat, J., Shi, X., Bashir, A., Ye, K., Devine, S. E., Talkowski, M., Mills, R. E., Marschall, T., Korbel, J., Eichler, E. E., and Lee, C. (2017b). Multi-platform discovery of haplotype-resolved structural variation in human genomes.

Chen, Z., Fu, B., Schweller, R., Yang, B., Zhao, Z., and Zhu, B. (2008). Linear time probabilistic algorithms for the singular haplotype reconstruction problem from snp fragments. *Journal of Computational Biology*, 15(5):535–546.

Chen, Z.-Z., Deng, F., and Wang, L. (2013). Exact algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 29(16):1938–1945.

Chin, C.-S., Alexander, D. H., Marks, P., Klammer, A. A., Drake, J., Heiner, C., Clum, A., Copeland, A., Huddleston, J., Eichler, E. E., et al. (2013). Nonhybrid, finished microbial genome assemblies from long-read smrt sequencing data. *Nature methods*, 10(6):563–569.

Chin, C.-S., Peluso, P., Sedlazeck, F. J., Nattestad, M., Concepcion, G. T., Clum, A., Dunn, C., O'Malley, R., Figueroa-Balderas, R., Morales-Cruz, A., et al. (2016). Phased diploid genome assembly with single-molecule real-time sequencing. *Nature methods*, 13(12):1050–1054.

Church, G. M. (2005). The personal genome project. *Molecular systems biology*, 1(1).

Cilibrasi, R., Iersel, L. v., Kelk, S., and Tromp, J. (2007). The Complexity of the Single Individual SNP Haplotyping Problem. *Algorithmica*, 49(1):13–36.

Collins, F. S., Morgan, M., and Patrinos, A. (2003). The human genome project: lessons from large-scale biology. *Science*, 300(5617):286–290.

Consortium, . G. P. et al. (2010). A map of human genome variation from population-scale sequencing. *Nature*, 467(7319):1061.

Consortium, . G. P. et al. (2015). A global reference for human genetic variation. *Nature*, 526(7571):68–74.

Consortium, E. P. et al. (2004). The encode (encyclopedia of dna elements) project. *Science*, 306(5696):636–640.

Consortium, I. H. et al. (2005). A haplotype map of the human genome. *Nature*, 437(7063):1299.

Cygan, M., Fomin, F. V., Kowalik, Ł., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., and Saurabh, S. (2015). *Parameterized algorithms*, volume 4. Springer.

Delaneau, O., Howie, B., Cox, A. J., Zagury, J.-F., and Marchini, J. (2013a). Haplotype estimation using sequencing reads. *The American Journal of Human Genetics*, 93(4):687–696.

Delaneau, O., Marchini, J., Consortium, . G. P., et al. (2014). Integrating sequence and array data to create an improved 1000 Genomes Project haplotype reference panel. *Nature communications*, 5.

Delaneau, O., Zagury, J.-F., and Marchini, J. (2013b). Improved whole-chromosome phasing for disease and population genetic studies. *Nature methods*, 10(1):5.

Deng, F., Cui, W., and Wang, L. (2013). A highly accurate heuristic algorithm for the haplotype assembly problem. *BMC genomics*, 14(2):S2.

Dinh, H. and Rajasekaran, S. (2011). A memory-efficient data structure representing exact-match overlap graphs with application for next-generation dna assembly. *Bioinformatics*, 27(14):1901–1907.

Duitama, J., Huebsch, T., McEwen, G., Suk, E.-K., and Hoehe, M. R. (2010). ReFHap: A reliable and fast algorithm for single individual haplotyping. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, BCB '10, pages 160–169, New York, NY, USA. ACM.

Eberle, M. A., Fritzilas, E., Krusche, P., Källberg, M., Moore, B. L., Bekritsky, M. A., Iqbal, Z., Chuang, H.-Y., Humphray, S. J., Halpern, A. L., et al. (2017). A reference data set of 5.4 million phased human variants validated by genetic inheritance from sequencing a three-generation 17-member pedigree. *Genome research*, 27(1):157–164.

Edge, P., Bafna, V., and Bansal, V. (2017). Hapcut2: robust and accurate haplotype assembly for diverse sequencing technologies. *Genome research*, 27(5):801–812.

Eid, J., Fehr, A., Gray, J., Luong, K., Lyle, J., Otto, G., Peluso, P., Rank, D., Baybayan, P., Bettman, B., et al. (2009). Real-time dna sequencing from single polymerase molecules. *Science*, 323(5910):133–138.

Eisenstein, M. (2015). Startups use short-read data to expand long-read sequencing market.

Falconer, E., Hills, M., Naumann, U., Poon, S. S., Chavez, E. A., Sanders, A. D., Zhao, Y., Hirst, M., and Lansdorp, P. M. (2012). Dna template strand sequencing of single-cells maps genomic rearrangements at high resolution. *Nature methods*, 9(11):1107–1112.

Fan, X., Chaisson, M., Nakhleh, L., and Chen, K. (2017). Hysa: a hybrid structural variant assembly approach using next-generation and single-molecule sequencing technologies. *Genome research*, 27(5):793–800.

Feige, U. (2014). Np-hardness of hypercube 2-segmentation. *arXiv preprint arXiv:1411.0821*.

Ferragina, P. and Manzini, G. (2000). Opportunistic data structures with applications. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 390–398. IEEE.

Fischer, S. O. and Marschall, T. (2016). Selecting reads for haplotype assembly. *biorxiv*, 046771.

Fouilhoux, P. and Mahjoub, A. R. (2012). Solving VLSI design and DNA sequencing problems using bipartization of graphs. *Computational Optimization and Applications*, 51(2):749–781.

Garg, S., Martin, M., and Marschall, T. (2016). Read-based phasing of related individuals. *Bioinformatics*, 32(12):i234–i242.

Garg, S. and Mömke, T. (2018). A qptas for gapless mec. *arXiv preprint arXiv:1804.10930*.

Garrison, E., Sirén, J., Novak, A. M., Hickey, G., Eizenga, J. M., Dawson, E. T., Jones, W., Lin, M. F., Paten, B., and Durbin, R. (2017). Sequence variation aware genome references and read mapping with the variation graph toolkit. *bioRxiv*, page 234856.

Giordano, F., Aigrain, L., Quail, M. A., Coupland, P., Bonfield, J. K., Davies, R. M., Tischler, G., Jackson, D. K., Keane, T. M., Li, J., et al. (2017). De novo yeast genome assemblies from minion, pacbio and miseq platforms. *Scientific reports*, 7.

Glusman, G., Cox, H. C., and Roach, J. C. (2014). Whole-genome haplotyping approaches and genomic medicine. *Genome Medicine*, 6(9):73.

Gnerre, S., MacCallum, I., Przybylski, D., Ribeiro, F. J., Burton, J. N., Walker, B. J., Sharpe, T., Hall, G., Shea, T. P., Sykes, S., et al. (2011). High-quality draft assemblies of mammalian genomes from massively parallel sequence data. *Proceedings of the National Academy of Sciences*, 108(4):1513–1518.

Gonnella, G. and Kurtz, S. (2012). Readjoiner: a fast and memory efficient string graph-based sequence assembler. *BMC bioinformatics*, 13(1):82.

Green, S. J., Monreal, R. P., White, A. T., Bayer, T. G., Green, S. J., Monreal, R. P., White, A. T., Bayer, T. G., Arquiza, Y. D., White, A. T., Green, S. J., Buenaflor, R., and Arquiza, J. N. Y. D. (1999). Phrap documentation.

Greenberg, H. J., Hart, W. E., and Lancia, G. (2004). Opportunities for combinatorial optimization in computational biology. *INFORMS Journal on Computing*, 16(3):211–231.

Grohme, M. A., Schloissnig, S., Rozanski, A., Pippel, M., Young, G. R., Winkler, S., Brandl, H., Henry, I., Dahl, A., Powell, S., et al. (2018). The genome of schmidtea mediterranea and the evolution of core cellular mechanisms. *Nature*, 554(7690):56.

Harrow, J., Frankish, A., Gonzalez, J. M., Tapanari, E., Diekhans, M., Kokocinski, F., Aken, B. L., Barrell, D., Zadissa, A., Searle, S., et al. (2012). Gencode: the reference human genome annotation for the encode project. *Genome research*, 22(9):1760–1774.

He, D., Choi, A., Pipatsrisawat, K., Darwiche, A., and Eskin, E. (2010). Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 26(12):i183–i190.

Hernandez, D., François, P., Farinelli, L., Østerås, M., and Schrenzel, J. (2008). De novo bacterial genome sequencing: millions of very short reads assembled on a desktop computer. *Genome research*, 18(5):802–809.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30.

Huddleston, J., Chaisson, M. J., Steinberg, K. M., Warren, W., Hoekzema, K., Gordon, D., Graves-Lindsay, T. A., Munson, K. M., Kronenberg, Z. N., Vives, L., et al. (2017). Discovery and genotyping of structural variation from long-read haploid genome sequence data. *Genome research*, 27(5):677–685.

Hunt, M., De Silva, N., Otto, T. D., Parkhill, J., Keane, J. A., and Harris, S. R. (2015). Circlator: automated circularization of genome assemblies using long sequencing reads. *Genome biology*, 16(1):294.

Idury, R. M. and Waterman, M. S. (1995). A new algorithm for dna sequence assembly. *Journal of computational biology*, 2(2):291–306.

Jackman, S. D., Vandervalk, B. P., Mohamadi, H., Chu, J., Yeo, S., Hammond, S. A., Jahesh, G., Khan, H., Coombe, L., Warren, R. L., et al. (2017). Abyss 2.0: resource-efficient assembly of large genomes using a bloom filter. *Genome research*, 27(5):768–777.

Jansen, T. (1998). Introduction to the theory of complexity and approximation algorithms. In *Lectures on Proof Verification and Approximation Algorithms*, pages 5–28. Springer.

Jeck, W. R., Reinhardt, J. A., Baltrus, D. A., Hickenbotham, M. T., Magrini, V., Mardis, E. R., Dangl, J. L., and Jones, C. D. (2007). Extending assembly of short dna sequences to handle error. *Bioinformatics*, 23(21):2942–2944.

Jiao, Y., Xu, J., and Li, M. (2004). On the k-closest substring and k-consensus pattern problems. In *CPM*, volume 3109 of *Lecture Notes in Computer Science*, pages 130–144. Springer.

Kajitani, R., Toshimoto, K., Noguchi, H., Toyoda, A., Ogura, Y., Okuno, M., Yabana, M., Harada, M., Nagayasu, E., Maruyama, H., et al. (2014). Efficient de novo assembly of highly heterozygous genomes from whole-genome shotgun short reads. *Genome research*, 24(8):1384–1395.

Kamath, G. M., Shomorony, I., Xia, F., Courtade, T. A., and David, N. T. (2017). Hinge: long-read assembly achieves optimal repeat resolution. *Genome research*, 27(5):747–756.

Kang, S.-H., Jeong, I.-S., Cho, H.-G., and Lim, H.-S. (2010). Hapassembler: A web server for haplotype assembly from snp fragments using genetic algorithm. *Biochemical and biophysical research communications*, 397(2):340–344.

Klau, G. W. and Marschall, T. (2017). A guided tour to computational haplotyping. In *Conference on Computability in Europe*, pages 50–63. Springer.

Kleinberg, J. M., Papadimitriou, C. H., and Raghavan, P. (1998). Segmentation problems. In *STOC*, pages 473–482. ACM.

Kleinberg, J. M., Papadimitriou, C. H., and Raghavan, P. (2004). Segmentation problems. *J. ACM*, 51(2):263–280.

Koren, S., Walenz, B. P., Berlin, K., Miller, J. R., Bergman, N. H., and Phillippy, A. M. (2017). Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation. *Genome research*, 27(5):722–736.

Kuleshov, V. (2014). Probabilistic single-individual haplotyping. *Bioinformatics*, 30(17):i379–i385.

Lancia, G., Bafna, V., Istrail, S., Lippert, R., and Schwartz, R. (2001). SNPs problems, complexity, and algorithms. In Heide, F. M. a. d., editor, *Algorithms ESA 2001*, number 2161 in Lecture Notes in Computer Science, pages 182–193. Springer Berlin Heidelberg.

Laszlo, A. H., Derrington, I. M., Ross, B. C., Brinkerhoff, H., Adey, A., Nova, I. C., Craig, J. M., Langford, K. W., Samson, J. M., Daza, R., et al. (2014). Decoding long nanopore sequencing reads of natural dna. *Nature biotechnology*, 32(8):829–833.

Lawler, E. L. (1979). Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356.

Levy, S., Sutton, G., Ng, P. C., Feuk, L., Halpern, A. L., Walenz, B. P., Axelrod, N., Huang, J., Kirkness, E. F., Denisov, G., et al. (2007). The diploid genome sequence of an individual human. *PLoS biology*, 5(10):e254.

Li, H. (2012). Exploring single-sample snp and indel calling with whole-genome de novo assembly. *Bioinformatics*, 28(14):1838–1844.

Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*.

Li, H. (2015a). Bfc: correcting illumina sequencing errors. *Bioinformatics*, 31(17):2885–2887.

Li, H. (2015b). Fermikit: assembly-based variant calling for illumina resequencing data. *Bioinformatics*, 31(22):3694–3696.

Li, H. (2016). Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110.

Li, M., Ma, B., and Wang, L. (2002). Finding similar regions in many sequences. *J. Comput. Syst. Sci.*, 65(1):73–96.

Li, Z., Chen, Y., Mu, D., Yuan, J., Shi, Y., Zhang, H., Gan, J., Li, N., Hu, X., Liu, B., et al. (2012). Comparison of the two major classes of assembly algorithms: overlap–layout–consensus and de-bruijn-graph. *Briefings in functional genomics*, 11(1):25–37.

Lieberman-Aiden, E., Van Berkum, N. L., Williams, L., Imakaev, M., Ragoczy, T., Telling, A., Amit, I., Lajoie, B. R., Sabo, P. J., Dorschner, M. O., et al. (2009). Comprehensive mapping of long-range interactions reveals folding principles of the human genome. *science*, 326(5950):289–293.

Lim, H.-S., Jeong, I.-S., and Kang, S.-H. (2012). Individual haplotype assembly of apis mellifera (honeybee) using a practical branch and bound algorithm. *Journal of Asia-Pacific Entomology*, 15(3):375–381.

Lin, Y., Yuan, J., Kolmogorov, M., Shen, M. W., Chaisson, M., and Pevzner, P. A. (2016). Assembly of long error-prone reads using de bruijn graphs. *Proceedings of the National Academy of Sciences*, 113(52):E8396–E8405.

Lippert, R., Schwartz, R., Lancia, G., and Istrail, S. (2002). Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in bioinformatics*, 3(1):23–31.

Loh, P.-R., Danecek, P., Palamara, P. F., Fuchsberger, C., Reshef, Y. A., Finucane, H. K., Schoenherr, S., Forer, L., McCarthy, S., Abecasis, G. R., et al. (2016a). Reference-based phasing using the haplotype reference consortium panel. *Nature genetics*, 48(11):1443.

Loh, P.-R., Palamara, P. F., and Price, A. L. (2016b). Fast and accurate long-range phasing in a uk biobank cohort. *Nature genetics*, 48(7):811.

Luo, R., Liu, B., Xie, Y., Li, Z., Huang, W., Yuan, J., He, G., Chen, Y., Pan, Q., Liu, Y., et al. (2012). Soapdenovo2: an empirically improved memory-efficient short-read de novo assembler. *Gigascience*, 1(1):18.

Ma, L., Xiao, Y., Huang, H., Wang, Q., Rao, W., Feng, Y., Zhang, K., and Song, Q. (2010). Direct determination of molecular haplotypes by chromosome microdissection. *Nature methods*, 7(4):299.

Maier, D. (1978). The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM)*, 25(2):322–336.

Marchini, J., Cutler, D., Patterson, N., Stephens, M., Eskin, E., Halperin, E., Lin, S., Qin, Z. S., Munro, H. M., Abecasis, G. R., and Donnelly, P. (2006). A comparison of phasing algorithms for trios and unrelated individuals. *American Journal of Human Genetics*, 78(3):437–450.

Martin, M., Patterson, M., Garg, S., Fischer, S. O., Pisanti, N., Klau, G. W., Schoenhuth, A., and Marschall, T. (2016). Whatshap: fast and accurate read-based phasing. *bioRxiv*, page 085050.

Matsumoto, H. and Kiryu, H. (2013). Mixsih: a mixture model for single individual haplotyping. *BMC genomics*, 14(2):S5.

Maxam, A. M. and Gilbert, W. (1977). A new method for sequencing dna. *Proceedings of the National Academy of Sciences*, 74(2):560–564.

Mazrouee, S. and Wang, W. (2014). Fasthap: fast and accurate single individual haplotype reconstruction using fuzzy conflict graphs. *Bioinformatics*, 30(17):i371–i378.

Medvedev, P., Georgiou, K., Myers, G., and Brudno, M. (2007). Computability of models for sequence assembly. In *WABI*, volume 4645, pages 289–301. Springer.

Medvedev, P., Pham, S., Chaisson, M., Tesler, G., and Pevzner, P. (2011). Paired de bruijn graphs: a novel approach for incorporating mate pair information into genome assemblers. *Journal of Computational Biology*, 18(11):1625–1634.

Melsted, P. and Pritchard, J. K. (2011). Efficient counting of k-mers in dna sequences using a bloom filter. *BMC bioinformatics*, 12(1):333.

Mitzenmacher, M. and Upfal, E. (2005). *Probability and Computing*. Cambridge.

Mostovoy, Y., Levy-Sakin, M., Lam, J., Lam, E. T., Hastie, A. R., Marks, P., Lee, J., Chu, C., Lin, C., Džakula, Ž., et al. (2016). A hybrid approach for de novo human genome sequence assembly and phasing. *Nature methods*, 13(7):587–590.

Motwani, R. and Raghavan, P. (2010). *Randomized algorithms*. Chapman & Hall/CRC.

Mousavi, S. R., Mirabolghasemi, M., Bargesteh, N., and Talebi, M. (2011). Effective haplotype assembly via maximum boolean satisfiability. *Biochemical and biophysical research communications*, 404(2):593–598.

Myers, E. W. (1995). Toward simplifying and accurately formulating fragment assembly. *Journal of Computational Biology*, 2(2):275–290.

Myers, E. W. (2005). The fragment assembly string graph. *Bioinformatics*, 21(suppl_2):ii79–ii85.

Myers, E. W., Sutton, G. G., Delcher, A. L., Dew, I. M., Fasulo, D. P., Flanigan, M. J., Kravitz, S. A., Mobarry, C. M., Reinert, K. H., Remington, K. A., et al. (2000). A whole-genome assembly of drosophila. *Science*, 287(5461):2196–2204.

Nagarajan, N. and Pop, M. (2009). Parametric complexity of sequence assembly: theory and applications to next generation sequencing. *Journal of computational biology*, 16(7):897–908.

Nagarajan, N. and Pop, M. (2013). Sequence assembly demystified. *Nature Reviews Genetics*, 14(3):157–167.

O'Connell, J., Gurdasani, D., Delaneau, O., Pirastu, N., Ulivi, S., Cocca, M., Traglia, M., Huang, J., Huffman, J. E., Rudan, I., McQuillan, R., Fraser, R. M., Campbell, H., Polasek, O., Asiki, G., Ekoru, K., Hayward, C., Wright, A. F., Vitart, V., Navarro, P., Zagury, J.-F., Wilson, J. F., Toniolo, D., Gasparini, P., Soranzo, N., Sandhu, M. S., and Marchini, J. (2014). A general approach for haplotype phasing across the full spectrum of relatedness. *PLoS Genet*, 10(4):e1004234.

Ono, Y., Asai, K., and Hamada, M. (2013). PBSIM: PacBio reads simulator–toward accurate genome assembly. *Bioinformatics*, 29(1):119–121.

Ostrovsky, R. and Rabani, Y. (2002). Polynomial-time approximation schemes for geometric min-sum median clustering. *J. ACM*, 49(2):139–156.

Paten, B., Novak, A. M., Garrison, E., and Hickey, G. (2017). Superbubbles, ultrabubbles and cacti. In *International Conference on Research in Computational Molecular Biology*, pages 173–189. Springer.

Patterson, M., Marschall, T., Pisanti, N., Iersel, L. v., Stougie, L., Klau, G. W., and Schönhuth, A. (2014). WhatsHap: Haplotype assembly for future-generation sequencing reads. In Sharan, R., editor, *Proceedings of the 18th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, number 8394 in Lecture Notes in Computer Science, pages 237–249. Springer International Publishing.

Patterson, M., Marschall, T., Pisanti, N., van Iersel, L., Stougie, L., Klau, G. W., and Schönhuth, A. (2015). WhatsHap: Weighted haplotype assembly for future-generation sequencing reads. *Journal of Computational Biology*, 22(6):498–509.

Pendleton, M., Sebra, R., Pang, A. W. C., Ummat, A., Franzen, O., Rausch, T., Stütz, A. M., Stedman, W., Anantharaman, T., Hastie, A., et al. (2015). Assembly and diploid architecture of an individual human genome via single-molecule technologies. *Nature methods*, 12(8):780–786.

Pevzner, P. A., Tang, H., and Waterman, M. S. (2001). An eulerian path approach to dna fragment assembly. *Proceedings of the National Academy of Sciences*, 98(17):9748–9753.

Pirola, Y., Zaccaria, S., Dondi, R., Klau, G. W., Pisanti, N., and Bonizzoni, P. (2015). HapCol: accurate and memory-efficient haplotype assembly from long reads. *Bioinformatics*, page btv495.

Porubsky, D., Garg, S., Sanders, A. D., Korbel, J. O., Guryev, V., Lansdorp, P. M., and Marschall, T. (2017). Dense and accurate whole-chromosome haplotyping of individual genomes. *Nature Communications*, 8(1):1293.

Porubský, D., Sanders, A. D., Wietmarschen, N. v., Falconer, E., Hills, M., Spierings, D. C. J., Bevova, M. R., Guryev, V., and Lansdorp, P. M. (2016). Direct chromosome-length haplotyping by single-cell sequencing. *Genome Res.*

Pryszcz, L. P. and Gabaldón, T. (2016). Redundans: an assembly pipeline for highly heterozygous genomes. *Nucleic acids research*, 44(12):e113–e113.

Rautiainen, M. and Marschall, T. (2017). Aligning sequences to general graphs in o (v+ me) time. *bioRxiv*, page 216127.

Remy, J. and Steger, A. (2009). Approximation schemes for node-weighted geometric steiner tree problems. *Algorithmica*, 55(1):240–267.

Rhee, J.-K., Li, H., Joung, J.-G., Hwang, K.-B., Zhang, B.-T., and Shin, S.-Y. (2016). Survey of computational haplotype determination methods for single individual. *Genes & Genomics*, 38(1):1–12.

Rice, E. S., Kohno, S., John, J. S., Pham, S., Howard, J., Lareau, L. F., O'Connell, B. L., Hickey, G., Armstrong, J., Deran, A., et al. (2017). Improved genome assembly of american alligator genome reveals conserved architecture of estrogen signaling. *Genome research*, 27(5):686–696.

Roach, J., Glusman, G., Hubley, R., Montsaroff, S., Holloway, A., Mauldin, D., Srivastava, D., Garg, V., Pollard, K., Galas, D., Hood, L., and Smit, A. (2011). Chromosomal haplotypes by genetic phasing of human families. *The American Journal of Human Genetics*, 89(3):382–397.

Rødland, E. A. (2013). Compact representation of k-mer de bruijn graphs for genome read assembly. *BMC bioinformatics*, 14(1):313.

Sanders, A. D., Falconer, E., Hills, M., Spierings, D. C., and Lansdorp, P. M. (2017). Single-cell template strand sequencing by strand-seq enables the characterization of individual homologs. *Nature Protocols*, 12(6):1151–1176.

Sanger, F., Nicklen, S., and Coulson, A. R. (1977). Dna sequencing with chain-terminating inhibitors. *Proceedings of the national academy of sciences*, 74(12):5463–5467.

Sedlazeck, F. J., Lee, H., Darby, C. A., and Schatz, M. C. (2018). Piercing the dark matter: bioinformatics of long-range sequencing and mapping. *Nature Reviews Genetics*, page 1.

Selvaraj, S., Dixon, J. R., Bansal, V., and Ren, B. (2013). Whole-genome haplotype reconstruction using proximity-ligation and shotgun sequencing. *Nature biotechnology*, 31(12):1111–1118.

Seo, J.-S., Rhie, A., Kim, J., Lee, S., Sohn, M.-H., Kim, C.-U., Hastie, A., Cao, H., Yun, J.-Y., Kim, J., et al. (2016). De novo assembly and phasing of a korean human genome. *Nature*, 538(7624):243–247.

Simpson, J. T. and Durbin, R. (2010). Efficient construction of an assembly string graph using the fm-index. *Bioinformatics*, 26(12):i367–i373.

Simpson, J. T. and Durbin, R. (2012). Efficient de novo assembly of large genomes using compressed data structures. *Genome research*, 22(3):549–556.

Simpson, J. T. and Pop, M. (2015). The theory and practice of genome sequence assembly. *Annual review of genomics and human genetics*, 16:153–172.

Simpson, J. T., Wong, K., Jackman, S. D., Schein, J. E., Jones, S. J., and Birol, I. (2009). Abyss: a parallel assembler for short read sequence data. *Genome research*, 19(6):1117–1123.

Sohn, J.-i. and Nam, J.-W. (2016). The present and future of de novo whole-genome assembly. *Briefings in bioinformatics*, 19(1):23–40.

Sović, I., Skala, K., and Šikić, M. (2013). Approaches to dna de novo assembly. In *Information & Communication Technology Electronics & Microelectronics (MIPRO), 2013 36th International Convention on*, pages 351–359. IEEE.

Steinberg, K. M., Schneider, V. A., Graves-Lindsay, T. A., Fulton, R. S., Agarwala, R., Huddleston, J., Shiryev, S. A., Morgulis, A., Surti, U., Warren, W. C., et al. (2014). Single haplotype assembly of the human genome from a hydatidiform mole. *Genome research*, 24(12):2066–2076.

Sudmant, P. H., Rausch, T., Gardner, E. J., Handsaker, R. E., Abyzov, A., Huddleston, J., Zhang, Y., Ye, K., Jun, G., Fritz, M. H.-Y., et al. (2015). An integrated map of structural variation in 2,504 human genomes. *Nature*, 526(7571):75–81.

Sutton, G. G., White, O., Adams, M. D., and Kerlavage, A. R. (1995). Tigr assembler: A new tool for assembling large shotgun sequencing projects. *Genome Science and Technology*, 1(1):9–19.

Tarhio, J. and Ukkonen, E. (1988). A greedy approximation algorithm for constructing shortest common superstrings. *Theoretical computer science*, 57(1):131–145.

Tewhey, R., Bansal, V., Torkamani, A., Topol, E. J., and Schork, N. J. (2011). The importance of phase information for human genomics. *Nature reviews. Genetics*, 12(3):215.

Todd, J. A. (1933). A combinatorial problem. *Studies in Applied Mathematics*, 12(1-4):321–333.

Trevisan, L. (2012). On khot's unique games conjecture. *Bulletin (New Series) of the American Mathematical Society*, 49(1).

Vaser, R., Sović, I., Nagarajan, N., and Šikić, M. (2017). Fast and accurate de novo genome assembly from long uncorrected reads. *Genome research*, 27(5):737–746.

Vazirani, V. V. (2013). *Approximation algorithms*. Springer Science & Business Media.

Venter, J. C., Adams, M. D., Myers, E. W., Li, P. W., Mural, R. J., Sutton, G. G., Smith, H. O., Yandell, M., Evans, C. A., Holt, R. A., et al. (2001). The sequence of the human genome. *science*, 291(5507):1304–1351.

Vinson, J. P., Jaffe, D. B., O'Neill, K., Karlsson, E. K., Stange-Thomann, N., Anderson, S., Mesirov, J. P., Satoh, N., Satou, Y., Nusbaum, C., et al. (2005). Assembly of polymorphic genomes: algorithms and application to ciona savignyi. *Genome research*, 15(8):1127–1135.

Walker, B. J., Abeel, T., Shea, T., Priest, M., Abouelliel, A., Sakthikumar, S., Cuomo, C. A., Zeng, Q., Wortman, J., Young, S. K., et al. (2014). Pilon: an integrated tool for comprehensive microbial variant detection and genome assembly improvement. *PloS one*, 9(11):e112963.

Wang, R.-S., Wu, L.-Y., Li, Z.-P., and Zhang, X.-S. (2005). Haplotype reconstruction from snp fragments by minimum error correction. *Bioinformatics*, 21(10):2456–2462.

Wang, T.-C., Taheri, J., and Zomaya, A. Y. (2012). Using genetic algorithm in reconstructing single individual haplotype with minimum error correction. *Journal of biomedical informatics*, 45(5):922–930.

Wang, Y., Feng, E., and Wang, R. (2007). A clustering algorithm based on two distance functions for mec model. *Computational biology and chemistry*, 31(2):148–150.

Warren, R. L., Sutton, G. G., Jones, S. J., and Holt, R. A. (2006). Assembling millions of short dna sequences using ssake. *Bioinformatics*, 23(4):500–501.

Weisenfeld, N. I., Kumar, V., Shah, P., Church, D. M., and Jaffe, D. B. (2017). Direct determination of diploid genome sequences. *Genome research*, 27(5):757–767.

Williams, A. L., Housman, D. E., Rinard, M. C., and Gifford, D. K. (2010). Rapid haplotype inference for nuclear families. *Genome biology*, 11(10):R108.

Wu, J., Wang, J., and Chen, J. (2013). A heuristic algorithm for haplotype reconstruction from aligned weighted snp fragments. *International journal of bioinformatics research and applications*, 9(1):13–24.

Wulff, S., Urner, R., and Ben-David, S. (2013). Monochromatic bi-clustering. In *ICML (2)*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 145–153. JMLR.org.

Xiao, C.-L., Chen, Y., Xie, S.-Q., Chen, K.-N., Wang, Y., Luo, F., and Xie, Z. (2016). Mecat: an ultra-fast mapping, error correction and de novo assembly tool for single-molecule sequencing reads. *bioRxiv*, page 089250.

Xie, M., Wang, J., and Chen, J. (2008). A model of higher accuracy for the individual haplotyping problem based on weighted snp fragments and genotype with errors. *Bioinformatics*, 24(13):i105–i113.

Xie, M., Wang, J., and Jiang, T. (2012). A fast and accurate algorithm for single individual haplotyping. In *BMC systems biology*, volume 6, page S8. BioMed Central.

Yang, H., Chen, X., and Wong, W. H. (2011). Completely phased genome sequencing through chromosome sorting. *Proceedings of the National Academy of Sciences*, 108(1):12–17.

Ye, C., Ma, Z. S., Cannon, C. H., Pop, M., and Douglas, W. Y. (2012). Exploiting sparseness in de novo genome assembly. In *BMC bioinformatics*, volume 13, page S1. BioMed Central.

Yue, J.-X., Li, J., Aigrain, L., Hallin, J., Persson, K., Oliver, K., Bergström, A., Coupland, P., Warringer, J., Lagomarsino, M. C., et al. (2017). Contrasting evolutionary genome dynamics between domesticated and wild yeasts. *Nature genetics*, 49(6):913–924.

Zhao, Y.-Y., Wu, L.-Y., Zhang, J.-H., Wang, R.-S., and Zhang, X.-S. (2005). Haplotype assembly from aligned weighted snp fragments. *Computational Biology and Chemistry*, 29(4):281–287.

Zheng, G. X., Lau, B. T., Schnall-Levin, M., Jarosz, M., Bell, J. M., Hindson, C. M., Kyriazopoulou-Panagiotopoulou, S., Masquelier, D. A., Merrill, L., Terry, J. M., et al. (2016). Haplotyping germline and cancer genomes with high-throughput linked-read sequencing. *Nature biotechnology*, 34(3):303–311.

Zimin, A. V., Puiu, D., Luo, M.-C., Zhu, T., Koren, S., Marçais, G., Yorke, J. A., Dvořák, J., and Salzberg, S. L. (2017). Hybrid assembly of the large and highly repetitive genome of aegilops tauschii, a progenitor of bread wheat, with the masurca mega-reads algorithm. *Genome Research*, 27(5):787–792.

Zook, J. M., Chapman, B., Wang, J., Mittelman, D., Hofmann, O., Hide, W., and Salit, M. (2014). Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. *Nat Biotechnol*, 32(3):246–251.

Appendices

Chapter A

Additional Details

A.1 Proof of Lemma 3.1

Proof. We show the claim by using a randomized argument. To this end, we assume that for each i, the rows from U_i and L_i are selected uniformly at random from $U_i \cap \tau(M)$ and $L_i \cap \tau(M)$ and the rows from U'_i and L'_i are selected uniformly at random from $U'_i \cap \tau'(M)$ and $L'_i \cap \tau'(M)$. We argue that for each column, the expected number of errors is at most a factor $(1 + O(\varepsilon))$ larger than in an optimal solution. Then the claim follows from linearity of expectation and the fact that there is a selection with at most the expected number of errors.

We consider the jth column of M. Let $c := \tau(M)_{*,j}$, but without rows that have an entry "–" in column j. Let $p := |\{i : c_i = 0\}|/|c|$ be the fraction of zeros in c. By swapping the zeros and ones we can assume w.l.o.g. that $p \geq 1 - p$, i.e., $p \geq 1/2$. Our assumption implies $\tau_j = 0$ and the optimal solution has $(1 - p)|c|$ errors within c.

The general idea of the proof is as follows. Suppose we would select exactly one row from $\tau(M)$ uniformly at random. Then with probability p, the algorithm has $(1-p)|c|$ errors in c and with probability $(1 - p)$ the number of errors is $p|c|$. Therefore the expected number of errors is $(p(1 - p) + (1 - p)p)|c| = 2p(1 - p)|c|$. We obtain the approximation ratio $2p(1 - p)|c|/((1 - p)|c|) = 2p$.

We will see that the approximation ratio improves with choosing several rows instead of a single one. Additionally, we have to handle the circumstance that we only sample from $U \cup L$ and ignore X.

There is a further issue regarding U. Let s be the smallest index such that U_s and c intersect, i.e., U_s is the first set with binary entries in column j. Then rows sampled for U_s may be located outside of c at positions with wildcards in column j. We avoid the complications caused by the wildcards by only considering classes U_i for $i > s$.

To summarize, c has at least εr selected entries and we ignore at most $2\varepsilon^2 r$ of these due to X and U_s. For each $i > s$, we sample $1/\varepsilon^3$ rows from U_i. Let c' be c restricted to $\bigcup_{i>s} U_i$ and let c'' be c restricted to L. Let $\hat{c}$ be c without U_s and X and let $\bar{c}$ be the part of c in $X \cup U_s$. For each i, let c'_i be the fraction of zeros of c' in U_i and c''_i the fraction of zeros of c'' in L_i.

For each $i \leq \ell$, we define p'_i to be the fraction of zeros c'_i and p''_i the fraction of zeros c''_i.

We define a random variables $Y'_{i,k}$ for each $s < i \leq 1/\varepsilon^2$ and $Y''_{i,k}$ for each $1 \leq i \leq 1/\varepsilon^2$. In both cases, $1 \leq k \leq 1/\varepsilon^3$. For each i, k, we pick an entry from c'_i (c''_i) uniformly at random. Then $Y'_{i,k}$ ($Y''_{i,k}$) is the value of the picked entry. For all i, k, $E[Y'_{i,k}] = 1 - p'_i$ and $E[Y''_{i,k}] = 1 - p''_i$. Observe that the $Y_{i,k}$ are independent Poisson trials. Let $Y' := \sum_{s<i,1\leq k\leq 1/\varepsilon^3} Y'_{i,k}$ and $Y'' := \sum_{i,1\leq k\leq 1/\varepsilon^3} Y''_{i,k}$. We want to use Chernoff bounds to control the probability to take the wrong decision. It is sufficient to consider Y' with $s = 1/\varepsilon^2 - 1$, since in all other cases the probabilities are amplified more. Observe that we do not have to consider smaller s because we are given a good SWC-instance and therefore there are no wildcards in L or L'.

Let $\mu' := E[Y']$. We analyze the ranges of μ' separately.

Case 1: Let us assume that $\mu' \in [0, 1/(2e\varepsilon^3)]$. We define $\delta' := 1/(2\mu'\varepsilon^3) - 1$. Using a multiplicative Chernoff bound (cf. Mitzenmacher and Upfal (2005)), we obtain

$$\Pr(Y' \geq 1/(2\varepsilon^3)) < \left(\frac{e^{\delta'}}{(1+\delta')^{(1+\delta')}}\right)^{\mu'} = \left(\frac{1}{1+\delta'}\right)^{\mu'} \left(\frac{e}{1+\delta'}\right)^{\mu'\delta'} \tag{A.1}$$

$$= (2\mu'\varepsilon^3)^{\mu'} (e \cdot 2\mu'\varepsilon^3)^{(1/(2\varepsilon^3)-\mu')} \tag{A.2}$$

Note that both terms of (A.2) are numbers between zero and one. If $\mu' < 1/\varepsilon$, the right term is smaller than $\varepsilon^4\mu'$. Otherwise the left term is smaller than $\varepsilon^4\mu'$

The range of μ' implies that the majority of entries in $\hat{c}'$ is zero. Recall that $\hat{c}'$ has an $\varepsilon^3\mu'$ fraction of zeros. The expected number of errors done by the algorithm is therefore at most $(1 - \varepsilon^4 \cdot \mu') \cdot (\varepsilon^3\mu') + \varepsilon^4 \cdot \mu' \cdot (1 - \varepsilon^3\mu') = (1+\varepsilon)\varepsilon^3\mu'$.

Case 2: Let us assume that $\mu' \in (1/(2e\varepsilon^3), 1/(2\varepsilon^3) - 1/\varepsilon^2]$. We use Hoeffding's inequality Hoeffding (1963) to analyze the range. To this end, we scale Y' and obtain $\bar{Y}' := \varepsilon^3 Y'$, which has values between zero and one. Then

$$\Pr(\bar{Y}' - E[\bar{Y}'] \geq \varepsilon) \leq e^{-2\varepsilon^2/\varepsilon^3} = e^{-2/\varepsilon}.$$

Since for sufficiently small ε, $e^{-2/\varepsilon} < \varepsilon/(2e) \leq \varepsilon^4\mu'$, again we obtain a $(1+\varepsilon)$-approximation in expectation.

All other ranges now follow immediately: For $\mu' \in (1/(2\varepsilon^3) - 1/\varepsilon^2, 1/(2\varepsilon^3)]$ every solution is a $(1 + O(\varepsilon))$-approximation and for larger μ' the majority of entries in $\hat{c}'$ is one. The analysis is analogous.

In order to combine Y' and Y'', we introduce a bias for Y' such that we count rows i for $s < i \leq \ell$ with a factor $(1-\varepsilon)/(\varepsilon - \varepsilon^2)$. Then

$$\bar{Y} := \frac{\bar{Y}' \cdot (\ell - s)(1-\varepsilon)/(\varepsilon - \varepsilon^2) + \bar{Y}'' \cdot \ell}{(\ell - s)(1-\varepsilon)/(\varepsilon - \varepsilon^2) + \ell}.$$

Then, using the union bound, setting $\sigma_j = 0$ for $\bar{Y} < 1/2$ and $\sigma_j = 1$ otherwise gives an expected $1 + O(\varepsilon)$ approximation within $\hat{c}$. Errors in $\bar{c}$ are either also errors in an optimal solution, or they contribute at most a factor $O(\varepsilon)$ to the total number of errors. Thus overall we obtain an approximation ratio $1 + O(\varepsilon)$ within c. The algorithm $\mathrm{SWC}_{\varepsilon^3}$ has at most the same approximation ratio, since the only difference is that we do not fix the $Y_{i,k}$ to be zero or one. Thus the random process used by the algorithm can only have a lower variance.

This finishes our analysis for $\tau(M)_{*,j}$. For $\tau'(M)_{*,j}$, the proof is analogous.

□

We introduced a small but easy to handle imprecision due to the assumption that we can choose exactly the same number of strings from each range.

A.2 A simplified DP for a single solution string.

We describe a dynamic program (DP) for a simplified setup with SWC-instances that consists of strings only from one of the two solution strings and the DP computes a single solution string.

Algorithm (SWC^σ). We first globally guess the value $|\tau(M)| =: r$, i.e., we run the algorithm for all possible values and keep the best outcome. The algorithm works in two phases. The first phase is an initialization.

We initialize *each* of cell $\zeta := D(B, C, T)$ with the value computed by $\mathrm{SWC}_{\varepsilon^3}$ with the following parameters. As U_i and L_i, we use the chunks C. Since we only consider one solution string, we do not have to fix r' or U'_i. In the execution of $\mathrm{SWC}_{\varepsilon^3}$, we use the selection T instead of trying all possible selections, i.e., T determines all $\tilde{U}_i$ and $\tilde{L}_i$ in the algorithm.

The value of ζ is the number of errors in B. The computed solution is σ_ζ. We update the cells the second phase as follows. Consider a DP cells $\zeta = D(B, C, T)$ and let Π be the set of all possible predecessors of ζ. Suppose that all cells in Π are updated already. (This is the case, if we consider cells ordered by increasing value a, breaking ties arbitrarily.)

We try all cells $\hat{\zeta} \in \Pi$ (with all of its parameters marked by $\hat{\cdot}$) and consider the block B from column $\text{end}_{\hat{b}+1}$ on, which we call $\tilde{B}$. We then run $\text{SWC}_{\varepsilon^3}$ with the parameters and selections from ζ for $\tilde{B}$. Let err be the number of errors of the solution in the rows a to $c-1$ of $\tilde{B}$. We concatenate the computed solution string to $\sigma_{\hat{\zeta}}$. The new value of ζ is $\min\{\zeta, \hat{\zeta} + \text{err}\}$. Overall, the value of ζ is the minimum value over all $\hat{\zeta} \in \Pi$.

We iterate this procedure until all cells are updated. It might happen, however, that we were not able to compute the entire solution yet. The reason is that valid DP cells as specified select a large number of rows, which may not be possible in the end. In order to finish the DP, we additionally consider special cells that are defined as before, but with $c = n$. Intuitively, we use these cells when only at most $1/\varepsilon^4$ rows of $\tau(M)$ are left. For these cells, our computation considers the optimal solution for the suffix of σ.

Lemma A.1. For SWC-instances M of Gapless-MEC with a restriction that M contains strings from only one of the two solution strings (σ or σ'), the above algorithm is a PTAS.

Proof. Since all binary strings are feasible solutions, our algorithm vacuously produces a valid solution. The number of different DP cells is polynomial in the instance size since the number of variables is a constant (depending on ε) and each variable has a polynomial range. All computations can be done in polynomial time. Therefore the overall running time of the algorithm is polynomial.

To analyze the quality of the computed solution, we partition $\tau(M)$ into ranges. Starting from the top-most row of $\tau(M)$, for each $i \geq 0$, the ith range Y_i contains the next $(\varepsilon^{2i} - \varepsilon^{(2i+2)})r$ rows of $\tau(M)$. To be consistent with properties needed in later proofs, we ensure that the first row of each Y_i is contained in $\tau(M)$ and thus we add the rows between Y_i and Y_{i+1} to Y_i. We note that if only a constant number of rows of $\sigma(M)$ are left, we can compute the partial solutions optimally and there are DP cells for exactly this purpose: there is a DP cell ζ_i such that the last at most $1/\varepsilon^4$ rows of $\tau(M)$ are located between a and c and Y_i contains exactly these rows. To keep a clean notation, in the following we implicitly assume that cells with constantly many rows of $\sigma(M)$ are handled separately.

The block B_0 contains the rows of Y_0 and the columns one to the end of the first row of $\tau(Y_0)$. For each $i > 0$, block B_i contains the rows of Y_i and Y_{i+1}. It contains the columns after those of B_{i-1} to the end of the first row of Y_i.

According to Definition 3.9, the remaining parameters for cells ζ_i lead to at least as good a solution as the following choice. The set C is chosen such that each block is the U and L part of an ε^2-trisection and the chunks are the subdivisions of the trisection (Definitions 3.3 and 3.4). The selections S are chosen in the same way as $\text{SWC}_{\varepsilon^3}$ would choose them.

We inductively show that the value of each ζ_i is at most a factor $(1+\varepsilon)$ larger than the number of errors of an optimal solution restricted to the considered prefix. For $i = 0$ we only consider B_0 and the invariant follows directly from Lemma 3.2.

Suppose now that $i \geq 0$ and for all $\tilde{i} < i$ the invariant is true. Then we consider ζ_i. Let $\tilde{B}_i$ be the part of B_i after B_{i-1}.

We apply Lemma 3.2 to compute the string σ^{ζ_i}. We obtain a $(1+\varepsilon)$ for the prefix covered by σ^{ζ_i} for the following reason. The part before $\tilde{B}_i$ was fixed, and by our induction hypothesis, independent of the rows considered in $\tilde{B}$ we already have a $(1+\varepsilon)$ approximation. The part of σ^{ζ_i} within $\tilde{B}_i$ gives a $(1+\varepsilon)$ approximation by the claim of Lemma 3.2.

We continue the induction until the entire string σ is determined. □

List of Figures

List of Tables

List of Algorithms