Title

Author

Thesis for obtaining the title of Doctor of Engineering
of the Faculty of Mathematics and Computer Science
of Saarland University

Saarbrücken
YYYY

Abstract

Abstract goes here...

Kurzfassung

Abstrakt geht hier...

Acknowledgements

Acknowledgements goes here...

Contents

Chapter **1**

Introduction

In this chapter, we introduce the fundamentals of genetics and DNA sequencing. Furthermore, we introduce the important research problem in the field of genetics, the diploid genome assembly (haplotyping). We will see how we can mathematically formualate the diploid assembly problem as a computer scientist. Then we provide a high level description of the main methods used in this problem. Thereafter, we describe the limits and challenges faced currently in this field. We finish this chapter by an outline of the thesis.

1.1 Genetics, DNA sequencing and haplotyping

Genetics study the amazing phenomenon, *life*, at its most basic level, and this makes the science of genetics tremendously important and incredibly fascinating. More precisely, genetics is the study of genes, genetic variation, and heredity in living organisms. The genetics controls what an organism looks like and how it works. It has applications in different fields such as medicine, animal breeding, agriculture, and biotechnology, etc.

Classically, there are two ways of looking at the science of genetics. At its molecular end, the availability of sequence information for genomic analysis, together with sophisticated techniques for gene editing and replacement, and analysis of gene expression patterns, provides powerful explaination about how the genes in organisms work. At its other extreme, a knowledge of genetics is fundamental to an understanding of how organisms, populations and species evolve. One of the most exciting developments in this direction, in the last few years, is the way in which these two extremes have begun to come together, through the integrative effort of using molecular techniques to the problems of development, evolution, and speciation. The modern biology tools (analysis of genomic sequences and bioinformatics) are most intelligently uses these genetic principles to answer some difficult biological questions ranging from evolution to complex diseases. In this way, genetics has a central role in modern biology and its impact in everyday life will continue to increase.

What is actually genetic information? In all living organisms, the genetic information is encoded in the form of the DNA molecule. The DNA molecule is a chain on which many bases are ordered in a linear sequence, the bases — A, T, G, and C — as the letters of a genetic alphabet. The whole information within the DNA molecule of an organism is called its genome. The genome is futher divided into chromosomes. The genomes have single (haploid), two (diploid) or higher ploidy (polyploids). In this thesis, we focus on *diploid* living organisms. For example, humans are diploids, consisting of two copies of each chromosome called as homologous chromosomes or *haplotypes* — one inherited from mother and other from father. There are differences between these two copies of each chromosome known as genetic variation.

In 2001, a major breakthrough happened in the scientific history is the sequencing of the human genome (Collins et al., 2003). *Sequencing* is the operation that consists of determining the bases sequence of a DNA molecule and to enode it for further analysis. For sequencing genomes, there exists several kind of sequencing technologies that holds the following common properties:

- They produce genomes in fragments or pieces called as “reads”.
- The location of fragments is unknown.
- The fragments contain errors.

All the sequencing technologies produce huge amounts of sequencing data. The major challenge with these sequencing datasets is that the genomic information is partial, not complete and erroneous. Therefore, no sequencing technology deliver the completely sequenced genome. Because of the double helix structure of the DNA molecule, both strands are present and sequenced. The strands are based complemented (A to T and C to G) and read in the opposite way. A genome containing ACCTGC therefore may present reads as CCTG or CAGG: CAGG being the reverse-complement of CCTG read from the opposite strand.

The major differences between the sequencing technologies are essentially the errors and the read-lengths. We define an error rate of a read by the ratio of the number of incorrectly sequenced bases by the size of the read. Therefore, the sequencing technologies can further be categorized into three categories:

- Short read sequencing: It includes Illumina/Solexa sequencing (Bentley et al., 2008), which is the broadly used technology. It produces shorter reads (hundreds of bases) with high accuracy ($\leq 1\%$). This technology presents an order of magnitude high throughput and cheap sequencing.
- Long read sequencing: It includes Single Molecule Real Time sequencing (PacBio) (Eid et al., 2009) and Nanopore sequencing (ONT)(Laszlo et al., 2014), produces very long sequences, up to hundreds kilo-bases. However, PacBio and ONT exhibit a very high error rate of up to 15% and 38% respectively. On the other hand, there is the 10x Genomics technology (Eisenstein, 2015) that produces synthetic long reads that relies on short-read Illumina platform.
- Single-cell sequencing: In includes Strand-seq sequencing that provides Illumina-style reads, along with directionality of DNA, in which each single strand of a DNA molecule is distinguished based on its 5’–3’ orientation (Falconer et al., 2012).

In addition to these properties, each method may show different biases due to the protocols employed. The high G/C content regions are less covered by the short read technologies and the read ends present high error rate (Aird et al., 2011; Dohm et al., 2008). Since each sequencing technologies come with its own advantages and disadvantages, the best way to generate accurate and full haplotypes is to intergate all the datasets.

What are the challenges to produce diploid genome from sequencing data? From all the existing sequencing technologies, the first challenge comes from the lack of information of the *genomic origin* of reads. This absence of context and the small size and error rates of the sequences obtained, relatively to the genome size, makes it difficult to use reads as such. Ideally, we would need the access to the underlying genomes in their entirety.

Since the beginning of sequencing of DNA molecules, genomes are produced by structuring and ordering reads information. Then these reconstructed genomes can be used as references. Reference genomes are the best insight we have about the one-dimensional organization of information in living cells. They give access not only to the gene sequences that lead to proteins, but also to flanking sequences that altogether impact the functioning of living beings (Consortium et al., 2004). They also reveal the inner organization of the genome such as genes relative positions or chromosomes structure. Helping understanding the genomes and organisms evolution, as well as how all the living is ruled by the encoded information. Besides, reference genomes can be seen as an entry point for biologists to use other kinds of data. For instance, they may add information about the known genes positions and functions to annotate the genome (Harrow et al., 2012). Furthermore, the reference genomes are also used to solve the read ordering problem. Traditionally, the sequencing reads have been aligned to the reference genome, which helps in identifying the origin of each read over the genome and therefore, provide their potential ordering.

From all sequencing technologies (except Strand-Seq), the second important challenge is the lack of the *haplotypic identity* of reads, meaning, the haplotype that the read comes from. To determine the assignment of each read to one of the homologous copies of chromosome (haplotype) is an important

question. Thus, from the biological point of view, the haplotyping (SIH) problem consists in the reassignment of each read to the original haplotype. Once we know this identity, it becomes easier to assemble the reads from each haplotype separately to further reconstruct the two genome sequences of diploid organisms. The process of reconstructing the diploid assemblies from sequencing reads is known as *diploid genome assembly* or *haplotyping*.

Importance of Haplotypes. These haplotypes are required in order to correctly understand allele-specific expression and compound heterozygosity, investigating the genetics of common diseases, and to perform population genetic analyses of admixture, migration and selection (Tewhey et al., 2011a; Glusman et al., 2014b). Furthermore, these 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.

Example 1.1. To illustrate the haplotyping problem for a single genome, we consider a small example in Figure 1.1. The example shows seven variants, the differences between two copies, covered by the sequencing reads. The alleles that a read supports are printed in white. The sequencing reads contain erroneous alleles are shown in red. The goal of *haplotyping* problem is the re-assignment of phases to the reads. It is shown in the middle panel with the colored bars representing the assignment of each read to green or purple haplotype. Finally, the reads from each haplotype are separately assembled together to output two haplotypes shown at the bottom in purple and green.

The large amounts of sequencing data, which is erroneous, is generated every day and extracting useful information from these datasets to understand the biology of diploid genomes is a challenging problem. One of the promises of the information technology era is that many such biological problems can now be solved rapidly by computers. The study of how to solve such problems in order to achieve the best possible goal, or objective, has created the field of *optimization*. The *optimization* problem is defined as: given an object $\mathfrak{x}$, find a solution such that an optimization creterion $\mathfrak{f}$ is minimized or maximized.

1.2 Haplotyping as Combinatorial Optimization problem

For the haplotyping problem, which is to know the haplotypic identity of each read, we consider the reads aligned to the reference genome. A read aligner maps the reads to the reference genome, ideally to positions with a high similarity score with the read. The number of reads whose alignment covers a position is known as the coverage. Furthermore, we have SNVs detected using different variant calling algorithms. In case of bi-allelic variants, that is, those for which two different alleles are known on two copies of chromosome, three genotypes are possible. One typically denotes the reference allele as 0 and the alternative one as 1. Using this notation, the two chromosomal copies either both carry the reference allele (genotype 0/0), (or alternative allele (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.

Given the variants and the alignments, the goal here is to phase the variants and further generate the diploid assemblies. The variants over the genome can be phased by using reads aligned to the reference genome. This process is known as *read-based phasing* or *haplotyping*.

Mathematically, the aligned reads to the variants is written in the form of a matrix called SNP matrix. The SNP matrix is illustrated in Figure 1.2 for an example given in Figure 1.1. The SNP matrix is $\mathcal{F} \in \{0, 1, -\}^{R \times M}$, 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 presence of sequencing and mapping errors makes the haplotype assembly problem a challenging task. This involves dealing in some way with sequencing and mapping errors and leads to a

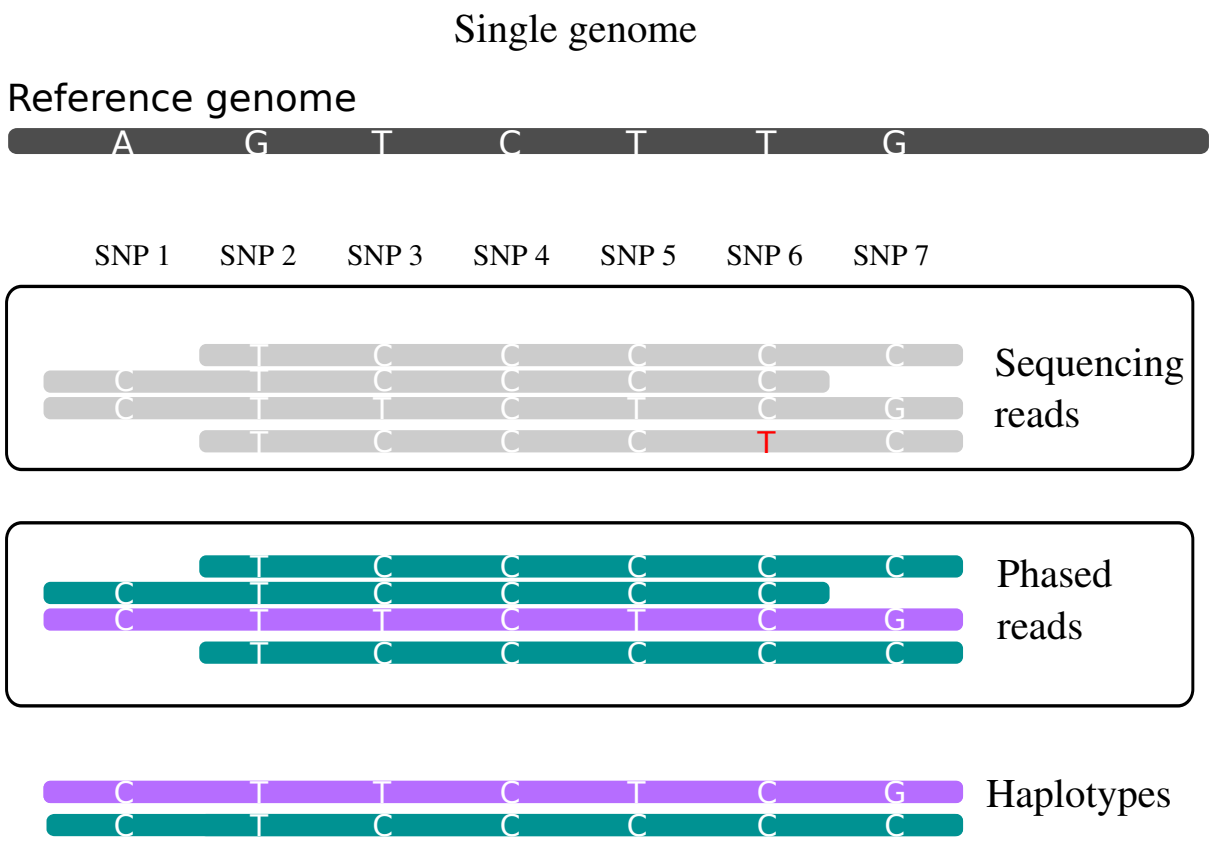


Figure 1.1: Seven variants covered by reads (horizontal bars) in a single individual. The alleles that a read supports are printed in white. The middle panel shows the phased reads in colors and haplotypes at the bottom over the seven variants.

computational task that is generally modeled as an optimization problem . In the literature, different combinatorial formulations of the problem have been proposed (Aguiar and Istrail, 2013; Dondi, 2012; Lippert et al., 2002). Among them, Minimum Error Correction (MEC) (Lippert et al., 2002) has been proved particularly successful in the reconstruction of accurate haplotypes for diploid species (Martin et al., 2016; He et al., 2010a; Chen et al., 2013b; Glusman et al., 2014a). It aims at correcting the input data with the minimum number of corrections to the SNP values, such that the resulting reads can be unambiguously partitioned into two sets, each one identifying a haplotype. To mathematically formulate the minimum number of corrections as an optimization problem, we require few definitions.

Definition 1.1 (Distance). 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 $\mathcal{F}$. Formally,

$$d(r_1, r_2) := \sum_{k=1}^{|M|} \left| \{ k \mid r_1(k) \neq - \ \wedge \ r_2(k) \neq - \ \wedge \ r_1(k) \neq r_2(k) \} \right|.$$

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

Now, we formulate the *haplotyping* as a Minimum Error Correction problem.

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

The MEC problem is NP-hard (Cilibrasi et al., 2007a). The weighted version of the problem associates a cost to every matrix entry. This is useful 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 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 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 $\mathcal{W}(j, k)$.

1.3 State-of-art approaches

There are mainly two kinds of approaches to diploid genome assembly: *reference guided* assembly and *denovo* assembly.

1.3.1 Reference guided assembly

The *reference guided* assembly consists in assembly of a diploid genome when we already have a reference for the species of the individual sequenced. We expect the target genome to be very close to the reference and we are interested to phase the differences of target to its reference. The usual pipeline to reference guided diploid assembly consists of the following steps:

- Align the reads to the reference genome.
- Detect variants based on aligned reads.
- Phase the variants using aligned reads.

For illustration, the toy example is given in Figure 1.1. The main focus of this study is on third point in the pipeline and we present the main algorithmic approaches to solve the haplotyping, both in theory and practice.

As we saw above, mathematically, the haplotyping using different types of sequencing datasets is formulated as MEC.

1.3.1.1 Theoretical guarantees

Broadly the MEC instances generated from different sequencing datasets are broadly divided into the following three types of instances:

- MEC: Instances where all the entries in $\mathcal{F}$ are $\{0, 1, -\}$. These instances can be generated by using Illumina, 10x Genomics and Strand-Seq sequencing datasets.
- Gapless-MEC: A MEC instance is called *gapless* if in each of the n rows of $\mathcal{F}$, all entries from $\{0, 1\}$ are consecutive. (As regular expression, a valid row is a word of length m from the language $-^*\{0, 1\}^*-^*$). These instances are generated from PacBio like technologies.
- Binary-MEC: Instances where all the entries in $\mathcal{F}$ are $\{0, 1\}$.

Example 1.2. To illustrate different types of MEC instances for a single genome, we consider a toy example in Figure 1.3. The example shows a mathemical representation of seven variants covered by reads. At the top, shown a general MEC instance consisting of arbitrary gaps and binary values, the middle shows 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 with no gaps.

Gapless-MEC is a generalization of a problem called Binary-MEC, the version of MEC with only instances M where all entries of M are in $\{0, 1\}$. Finding an optimal solution to Binary-MEC is equivalent to solving the hypercube 2-segmentation problem (H2S) which was introduced by Kleinberg, Papadimitriou, and Raghavan (Kleinberg et al., 1998, 2004) and which is known to be NP-hard (Feige, 2014; Kleinberg et al., 2004). The optimization version of Binary-MEC differs from H2S in that we minimize the number of mismatches instead of maximizing the number of matches. Binary-MEC

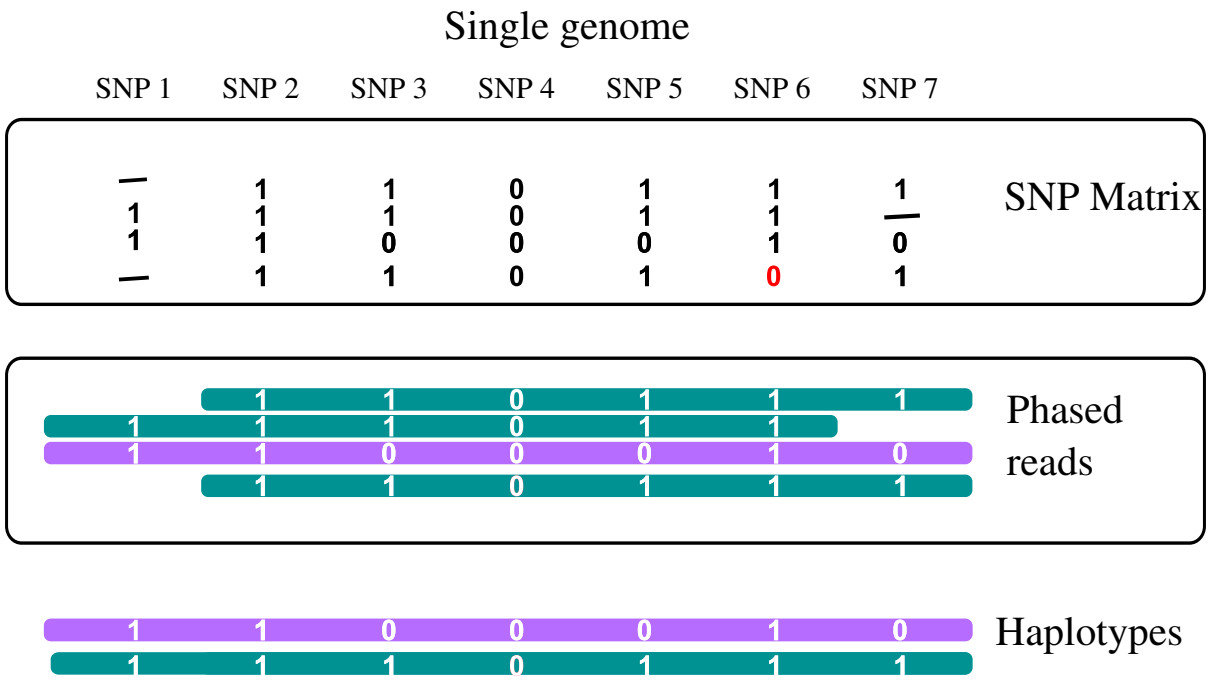


Figure 1.2: Example shows the SNP matrix for the example shown in Fig. 1.1. Seven variants covered by reads (horizontal bars) in a single individual. The allele in read is encoded as 1 if it matches the allele in the reference position at that position. The middle panel shows the phased reads in colors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

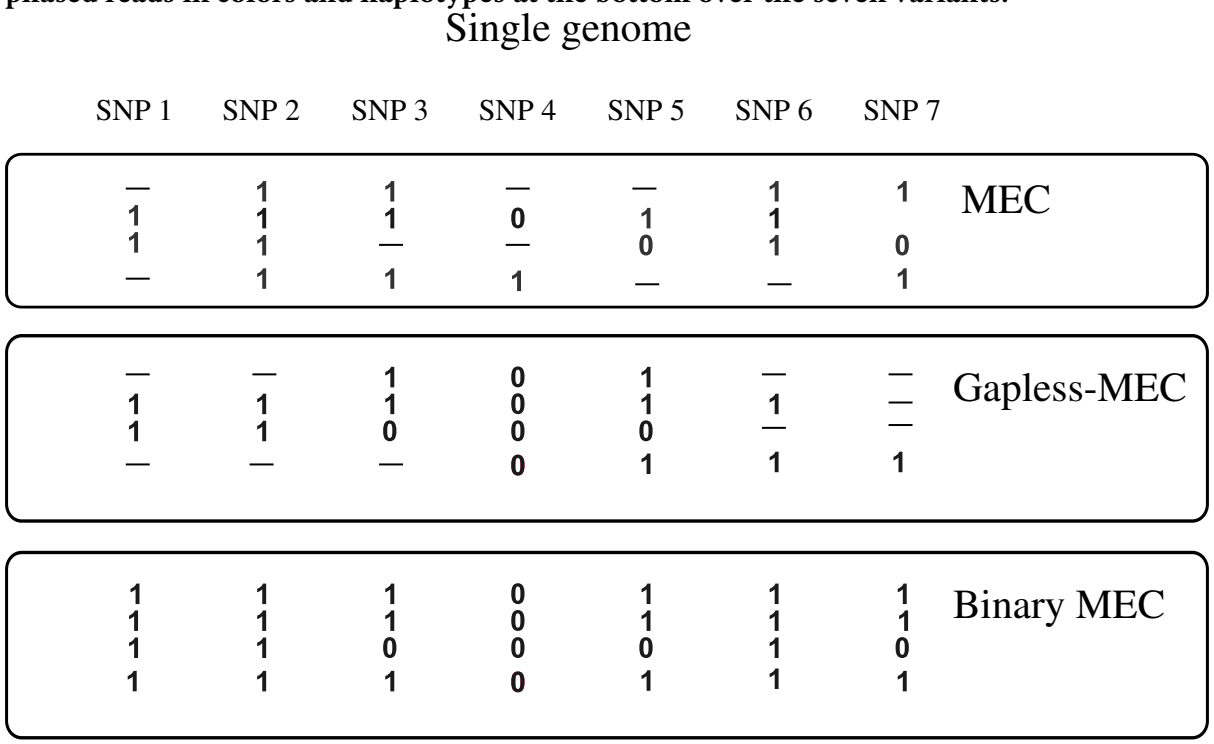


Figure 1.3: Seven variants covered by reads (horizontal bars) in a single individual are represented as MEC instances. At the top is a general MEC instance with arbitrary gaps, the middle is a Gapless-MEC instance with gaps only at its two ends and the bottom is a Binary-MEC instance which consists of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

allows for good approximations. Ostravsky and Rabiny (Ostrovsky and Rabani, 2002) obtained a PTAS for Binary-MEC based on random embeddings. Building on the work of Li et al. (Li et al., 2002), Jiao et

al. (Jiao et al., 2004) presented a deterministic PTAS for Binary-MEC.

Gapless-MEC was shown to be NP-hard by Cilibrasi et al. (Cilibrasi et al., 2007b).[1] Additionally, they showed that allowing a single gap in each strings renders the problem APX-hard. More recently, Bonizzoni et al. (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. To our knowledge, previous to our result their logarithmic factor approximation was also the best known approximation algorithm for Gapless-MEC.

Open Problem 1.1. The approximation status of Gapless-MEC is an open problem, Gapless-MEC instances are very important and are often produced by long-read single-ended PacBio or ONT technologies while haplotyping. To derive the polynomial time approximation algroithms for Gapless-MEC instances helps to solve these instances in a polynomial time even in practice.

1.3.1.2 Practical approaches

The approaches that works well in practice to perform phasing using sequencing datasets are broadly categorized into the following two categories:

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

Interger Linear Programming (ILP) is a mathemical optimization or feasibility in which some or all of the variables are restricted to be integers. Additionally, objective function and the constraints (other than the integer constraints) are linear. An ILP in standard form is 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 $\mathbf{c}, \mathbf{b}$ are vectors and A is a matrix, where all entries are integers.

For the haplotyping problem, we want to compute the optimal pair (h^0, h^1) for $\mathcal{F}$. For its MEC formulation, in (Chen et al., 2013c) algorithm, the binary variable x_k for column $\mathcal{F}(k)$ is considered such that its value is supposed to be 1 if and only if the $k^t h$ bit of h^0 and h^1 is 1 and 0 respectively. Moreover, the binary variable y_j for row $\mathcal{F}(j)$ is 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 constraints on all the binary variables for all the rows and columns is that the binary variables should belong to 0 or 1. The objective function is to minimize the number of flips in each entry from all the rows such that all rows become consistent to the original haplotypes h^0 or h^1. By using some auxiliary variables, we get a linear program, that gives an optimal solution to the problem.

Branch-and-Bound algorithm. Based on the haplotype definition in the previous section, haplotype determination can be viewed as a way to find the optimal path using a binary tree, because the problem can be converted into choosing the side between haplotypes h^0 and h^1. Li et al. (2005) tried to apply a so-called branch and bound algorithm. Each node is a fragment in the tree structure and the edge indicates the index of the haplotype group. From the root node, that is the first fragment, the algorithm adds a fragment and measures the MEC score. Then if the calculated score is bigger than the previous score, it would be divided. The branch and bound algorithm can identify the exact optimal solution, but the time complexity is exponentially increased by the number of fragments. Therefore, its use in a large-scale datasets is difficult.

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

Parameterized algorithms. Most of the exact algorithms to solve NP hard problems, if input parameters are not fixed; require time that is exponential (or at least superpolynomial) in the total size of the input. However, some problems can be solved by algorithms that are exponential only in the size of a fixed parameter while polynomial in the size of the input. Such an algorithm is called a fixed-parameter tractable (fpt-)algorithm, because the problem can be solved efficiently for small values of the fixed parameter.

Based on NGS data analysis, there are several parameters, such that read-length, coverage and sequencing errors, that can help in solving genomics problems efficiently. The art of choosing parameter that is small enough to work in practice is crucial. In the works by He et al. (2010a); Patterson et al. (2015a); Pirola et al. (2015a), different parameters are proposed to solve the haplotyping problem using NGS data. We will discuss these parameterized algorithms for haplotyping more in detail in Section(**[TODO: Section]**).

Additionally, it is shown that the parameterized algorithms for haplotyping, works well in practice for different types of sequencing datasets (Martin et al., 2016). The next obvious question is to develop a parameterized algorithm by integrating all sequencing datasets. To decide the parameter, that works for integrative sequencing datasets is very important.

Open Problem 1.2. To integrate all types of sequencing datasets to get accurate and complete haplotypes, the parameterized algorithm for haplotyping is an open problem.

Example 1.3. To illustrate the motivation to combine all the sequencing, the corresponding MEC instance for a single genome is shown in Figure 1.3. The example shows mathemical representation of reads covering the variants from different technologies such as Illumina, PacBio and Strand-Seq. At the top, shown a general MEC instance consisting of Illumina reads with gaps at ends and in the middle, the middle shows a Gapless-MEC instance with gaps only at its two ends and the bottom is a MEC instance which consists of only binary values with arbitray gaps.

Heuristic approaches A heuristic algorithm is one that is designed to solve a problem in a faster and more efficient fashion than traditional methods by sacrificing optimality, accuracy, precision, or completeness for speed. Heuristics can produce a solution individually or be used to provide a good baseline and are supplemented with optimization algorithms. Heuristic algorithms are most often employed when approximate solutions are sufficient and exact solutions are necessarily computationally expensive.

Clustering algorithms. In the work by (Wang et al., 2007), a clustering algorithm 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. This distance takes into account only the number of mismatches between two fragments. The second distance D' also takes into account the number of matches between the two fragments. This means 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.

1. for each possible pair of fragments in the SNP matrix the generalized Hamming distance is computed. Let r_1 and r_2 be the two furthest fragments according to Hamming distance, the two sets are initialized as $C1 = r_1$ and $C2 = r_2$.
2. Let H1 and H2 be the two consensus strings derived from C1 and C2: all the fragments are compared with H1 and H2 and assigned to the corresponding closer set. If a fragment is equidistant from the two consensus strings, the distance D' is used to decide to which set assign the fragment.
3. Once all fragments are assigned, the consensus strings H1 and H2 are updated and the algorithm restarts from (2). The procedure loops until a stable haplotype pair is found (i.e. when the consensus haplotypes are the same before and after the update).

Max-Cut based algorithm. HapCUT (Bansal and Bafna, 2008) approaches the haplotype assembly as a MAX-CUT problem. Given a certain haplotype pair H, a graph $G(H)$ is constructed such that there is a vertex for each column of the matrix $\mathcal{F}$ and there is an edge between two vertices of $G(H)$ if the

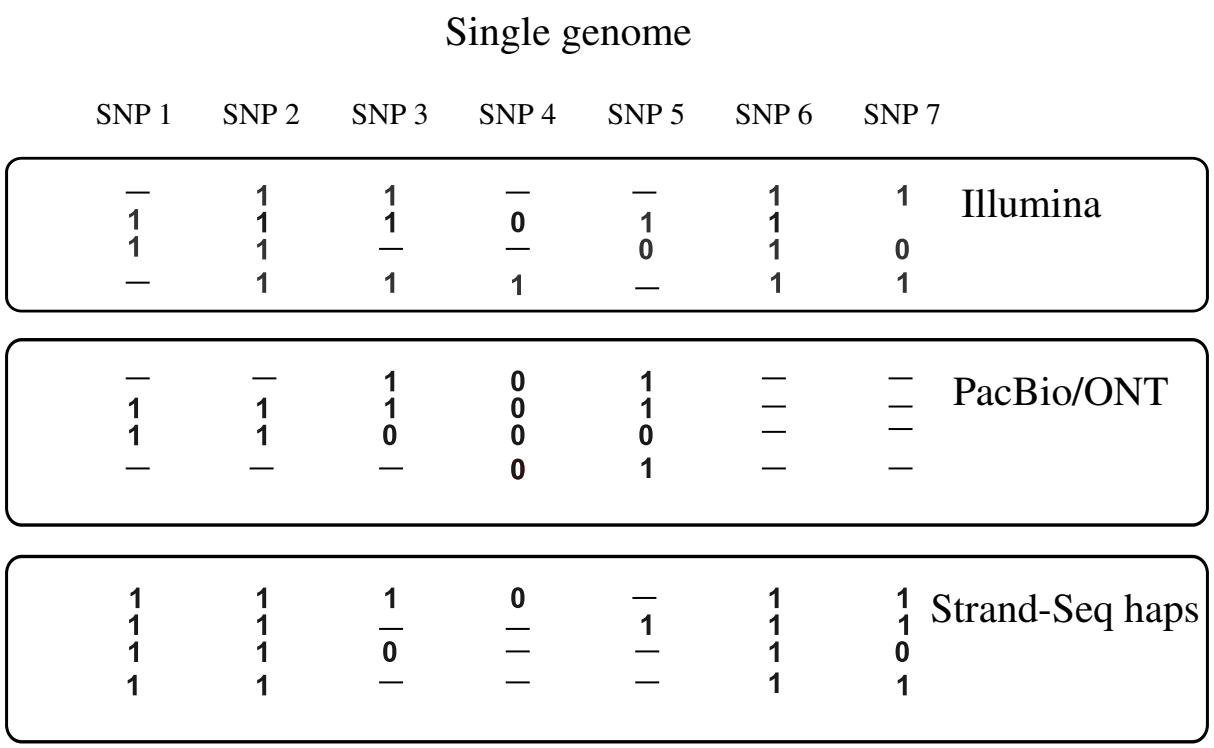


Figure 1.4: Seven variants covered by reads (horizontal bars) in a single individual are represented as MEC instances from different sequencing technologies.

corresponding columns in $\mathcal{F}$ are linked by at least one fragment. Consider the fragment $\mathcal{F}(j)$ such that it covers both positions k1 and k2. Let $\mathcal{F}(j)[k1, k2]$ and H[k1, k2] represent the restriction of $\mathcal{F}(j)$ and H to loci k1 and k2. There are two cases: $\mathcal{F}(j)[k1, k2]$ matches one of the two haplotype strings of H[k1, k2], or $\mathcal{F}(j)[k1, k2]$ does not match any. The weight w (k1, k2) associated with the edge between node k1 and k2 in the graph $G(H)$ is given by the number of fragments such that $\mathcal{F}(j)[k1, k2]$ does not match any string in H[k1, k2] minus the number of fragments such that the match exists. The higher w(k1, k2), the weaker is the correlation between the haplotype pair H and the SNP matrix restricted to columns k1 and k2. Let (S, $\mathcal{F}$ - S) be a cut of G, the weight of the cut is defined as follows:

$$w(S) = \sum_{j \in S, k \in \mathcal{F}-S} w(j, k)$$

Consider the haplotype pair H_S derived from H by flipping all the elements involved in S. It is shown that the problem of finding a haplotype pair minimizing the MEC score is reduced to the problem of finding a max-cut in $G(H)$. To solve max-cut problem, HapCut initializes the random haplotype pair, and then iteratively attempts to refine the haplotype pair to reduce MEC score till it is no longer possible to further reduce it.

The reference-based assembly has few disadvanatges. First because of the biases that the method present. We make the prior hypothesis that the genome to assemble is very close to the reference. This may mislead the assembly onto something too similar to the reference. Secondly the method is obviously not self-sufficient since a reference needs a prior reference to be constructed. For these reasons, we additionally consider the *denovo* (without reference) assembly.

1.3.2 Denovo diploid genome assembly

Instead of reference genome, the reads itself from the genome are used to construct the assembly graph, which is used as a backbone for phasing. The main challenges to construct the assembly graphs are that the genomes are very long and repeat-rich. Reads are very short and may contain errors and biases. For assembling the genomes, the assembly graphs can be categorized into two families: de Bruijn graph and overlap graph.

Overlap Layout Consensus The overlap layout consensus paradigm core notion is the overlap graph. The objective is to know how all reads can be positioned in relation to each others, to represent those connections in a graph and to consider all overlaps (not only maximal ones) to produce a solution. We know how reads can be ordered by knowing how they overlap. The overlap graph is a graph where reads are nodes, connected if they overlap significantly as illustrated in Figure 1.5b. The algorithm can be outlined by:

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

The first OLC assembler was Celera (Myers et al., 2000) and was designed to handle Sanger sequences. Celera uses a BLAST-like approach to compare each read to the others and to find significant overlaps. Then it compacts the overlaps presenting no ambiguity and tries to apply heuristics on the complex cases involving repeats. The final sequences are produced via a consensus to remove most sequencing errors. The complex repititive regions are hard to resolve, this results into fragmented assemblies constituted of consensus sequences that are supposed to be genome substrings. We call those sequences "contigs" for contiguous consensus sequence.

The overlap graphs can be simplified to string graphs by the transitive reduction of edges.

This paradigm was used a lot with long Sanger sequences and for relatively small genomes. Because of the cost of the pairwise overlaps computation, the OLC is too time consuming on high number of short reads from NGS. Thus, other solutions had to be found to be able to deal with the amount of reads to assemble large genomes.

De Bruijn graphs The de Bruijn graph (Figure 1.5a) is a directed graph representing overlaps between sequences of symbols, named after Nicolass Govert de Bruijn (Todd, 1933). Given an alphabet σ of m symbols, a k dimensional de Bruijn graph has the following properties.

1. m^k vertices produced by all words of length k from the alphabet σ
2. Two vertices A and B are connected by an edge from A to B if and only if the k - 1 suffix of A is equal to the k - 1 prefix of B.

The first application of the de Bruijn graph in genome assembly was introduced into the EULER assembler (Pevzner et al., 2001) in order to tackle assembly complexity. The idea was to consider a partial de Bruijn graph on the alphabet (A,C,T,G) constructed only with the vertices whose words of length k, called k-mers, appeared in the sequencing data. The intuition of this approach is the following:

1. A read is represented as a path in the graph
2. Reads that overlap with more than k nucleotides will share some k-mers
3. Extracting paths of such graph will produce assembled reads

De Bruijn graph and overlap graph The de Bruijn graph theoretically achieves the same tasks than the overlap graph, while being conceptually simpler and much more efficient for the three reasons detailed in the following:

- No alignment
- Abstracted coverage
- No consensus

The de Bruijn graph became widely used when the shorts reads from NGS appeared, as it was better suited than the OLC to handle this kind of sequencing data. The OLC approach did not scale well on the high number of sequences generated by NGS. The use of the de Bruijn graph is very interesting for short read assembly for its ability to deal with the high redundancy of such sequencing in a very efficient way. Indeed a kmer presents dozens of time in the sequencing dataset appears only once in the graph. This makes the de Bruijn graph structure not very sensible to the high coverage, unlike the OLC. The de Bruijn graph was first proposed as an alternative structure (Pevzner et al., 2001) because it was

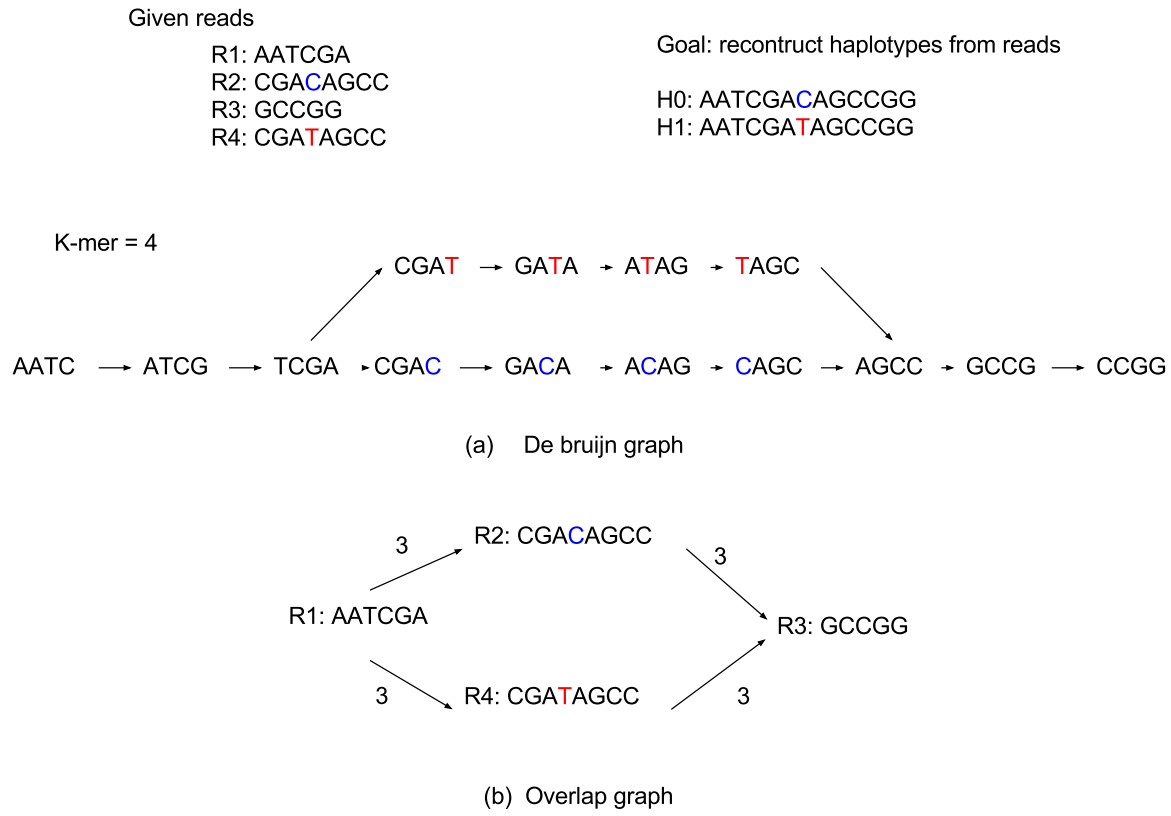


Figure 1.5: Figure shows the reads and reconstructed haplotypes using two graph approaches: (a) debruijn graph and (b) overlap graph.

less sensible to repeats. Repeats that were problematic in the OLC, creating very complex and edges heavy zones, are collapsed in the de Bruijn graph.

Graph structures There are mainly two types of structures in the above mentioned graphs that represent the genetic variation present over the genome.

Bubbles. The bubbles are defined as a set of disjoint paths that share the same start and end nodes. The bubbles in the graph structures represent the heterozygosity in the diploid organisms. The bubbles can contain simple SNVs with only one allele difference, or even large complex structural variations in the order of few tens of bases. Figure 1.5 illustrates how the bubbles in an assembly graph can contain both small variants (SNPs and indels up to several dozen base-pairs in length) and larger structural variants.

Repeats. The repeats over the genomes causes branches or cycles in the assembly graphs and, therefore, makes graph more complex and break its linear chain properties. This is illustrated in Figure 1.6. In this example, the repeat R causes cycle and branches in the graph. The assemblers generally handle these repeats by making a 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. Furthermore, the maximal repeat resolution is decided based on the read-length. If there is a read that is long enough to span the repeat region, we say the repeat is resolvable. Therefore, upcoming long-read sequencing technologies are a step forward to get better repeat resolved haplotypes.

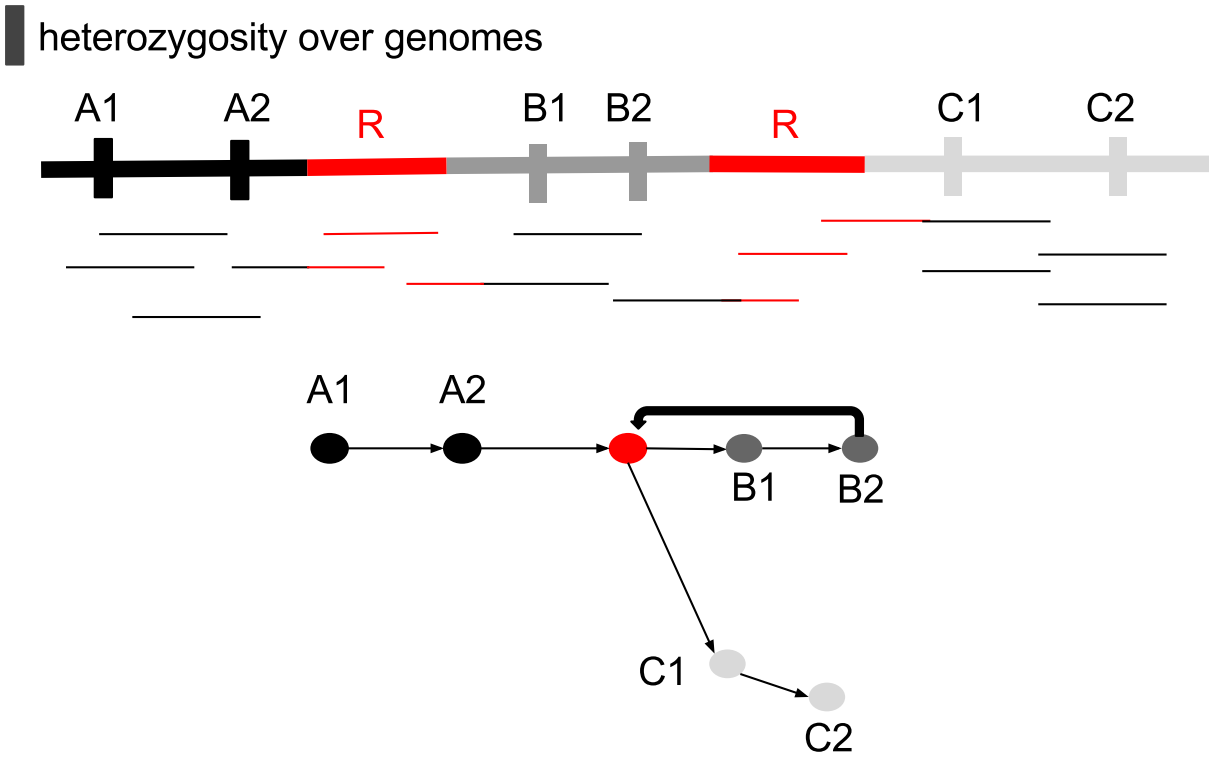


Figure 1.6: At the top, shown are the heterozygosity (in vertical bars) and repetitive regions (in red) over the genome. At the bottom, shown is the graph with nodes as heterozygous or repetitive region, and connections are based on the successive read overlap. The graph has cycles because of repetitive region shown in R, and further also causes two branches.

1.3.2.1 Haplotyping using assembly graph as reference

In the most recent method for diploid genome assembly, Falcon Unzip method (Chin et al., 2016) uses the PacBio based assembly graph. Falcon Unzip is a diploid-aware long-read assembler to assemble haplotype contigs or "haplotigs" that represent the diploid genome with correctly phased homologous chromosomes. The pipeline is given in Figure 1.7. Falcon Unzip begins by using reads to construct a string graph that contains sets of "haplotype-fused contigs" as well as bubbles representing divergent regions between homologous sequences (Fig. 1.7a). Next, FALCON-Unzip identifies read haplotypes using phasing information from heterozygous positions that it identifies (Fig. 1.7b). Phased reads are then used to assemble haplotigs and primary contigs (backbone contigs for both haplotypes) (Fig. 1.7c) that form the final diploid assembly with phased single-nucleotide polymorphisms (SNPs) and structural variants (SVs).

Phasing using primary contigs. In Falcon Unzip, the reads are aligned to primary contigss and heterozygous SNPs (het-SNPs) are called by analyzing the base frequency of the detailed sequence alignments. A simple phasing algorithm was developed to identify phased SNPs. Along each contig, the algorithm assigns phasing blocks where "chained phased SNPs" can be identified. Within each block, if a raw read contains a sufficient number of het-SNPs, it assigns a haplotype phase for the read unambiguously. Combined with the block and the haplotype phase information, it assigns a "block-phase" tag for each phased read in each phasing block. Some reads might not have enough phasing information. 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. The initial assembly graph is fused using phased reads and the haplotigs are generated in a greedy manner using local conservative approach.

To our knowledge, there is no work that phases directly from the assembly graph.

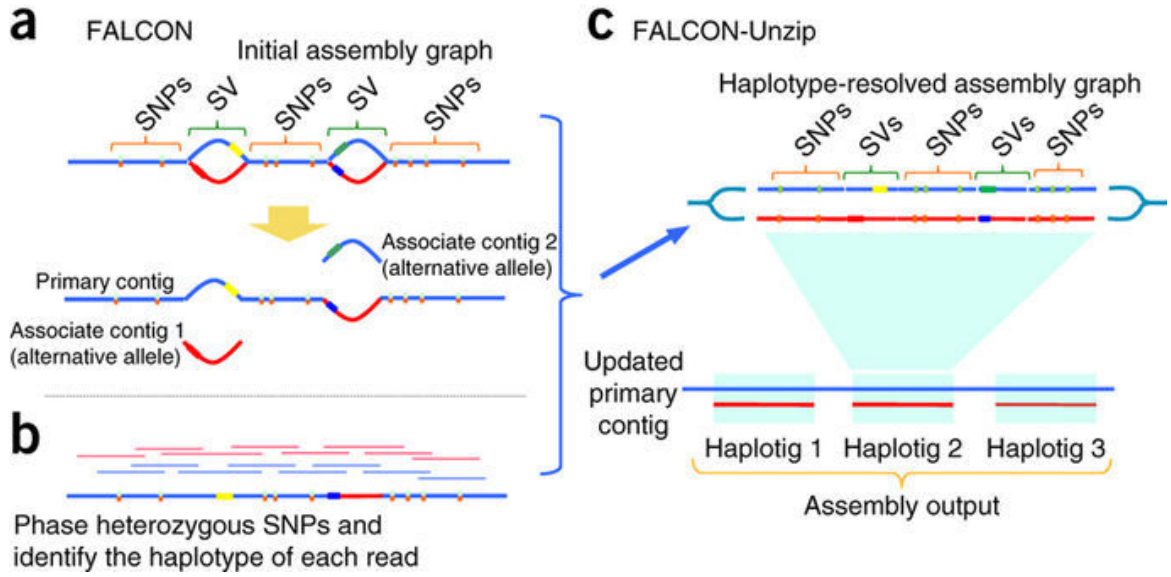


Figure 1.7: (a) An initial assembly is computed by FALCON, which error corrects the raw reads (not shown) and then assembles them using a string graph of the read overlaps. The assembled contigs are further refined by FALCON-Unzip into a final set of contigs and haplotigs. (b) Phase heterozygous SNPs and group reads by haplotype. (c) The phased reads are used to open up the haplotype-fused path and generate as output a set of primary contigs and associated haplotigs.

Open Problem 1.3. Phasing bubbles directly from the assembly graph is an open problem. Additionally, the hybrid of multiple sequencing technologies such that using one technology that produces shorter, more accurate reads, and a second technology that delivers long reads, can lead to better quality haplotypes.

Phasing using an assembly graph has several advantages over other approaches. For example, it is possible to phase larger blocks at once, because paths in an assembly graph can span multiple variants. Moreover, it is easier to detect large structural variants, such as translocations and other rearrangements, in an assembly graph. The graph-based approach provides a way to both accurately detect all types of structural variation and perform further downstream analyses.

Example 1.4. Figure 1.8 demonstrates the advantages of graph-based approach over contig-based 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 reads r_3 and r_4 because they cover less than two SNVs, resulting in fragmented haplotigs. In contrast, the graph-based approaches attempt to detect all types of SVs and can phase all the reads covering these SVs. In this example, reads r_3 and r_4 can also be phased because they cover SV1 and SV2, producing end-to-end haplotigs.

Therefore, the graph-based approaches are powerful to deliver more complete and contiguous haplotigs.

1.3.3 Pedigree of genomes

There is an another approach to haplotyping when we have sequencing datasets from the families of genomes. In this, we can take advantage of two things: one is the sequencing data itself of each individual and the other is the principles of the Mendelian segregation of alleles in pedigrees. There are alleles that are specific to a single founding chromosome within a pedigree, which we refer to as lineage-specific alleles, are highly informative for identifying haplotypes that are identical-by-decent

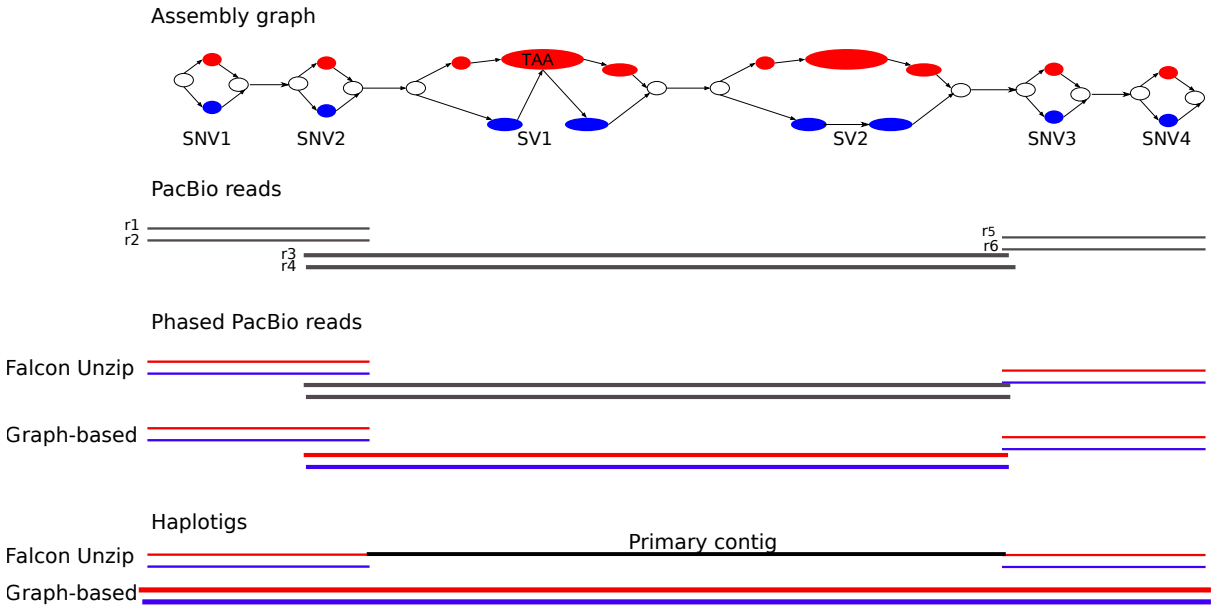


Figure 1.8: Input: an assembly graph (top) (consisting of four SNVs and two SVs) and the PacBio reads $r_1, r_2, r_3, r_4, r_5, r_6$ (gray). Output: the phased reads (colored in blue and red) and haplotigs (bottom) using Falcon Unzip and our graph-based approach. Our graph-based phase all the reads, contrarily, Falcon Unzip don't phase the reads r_3 and r_4,

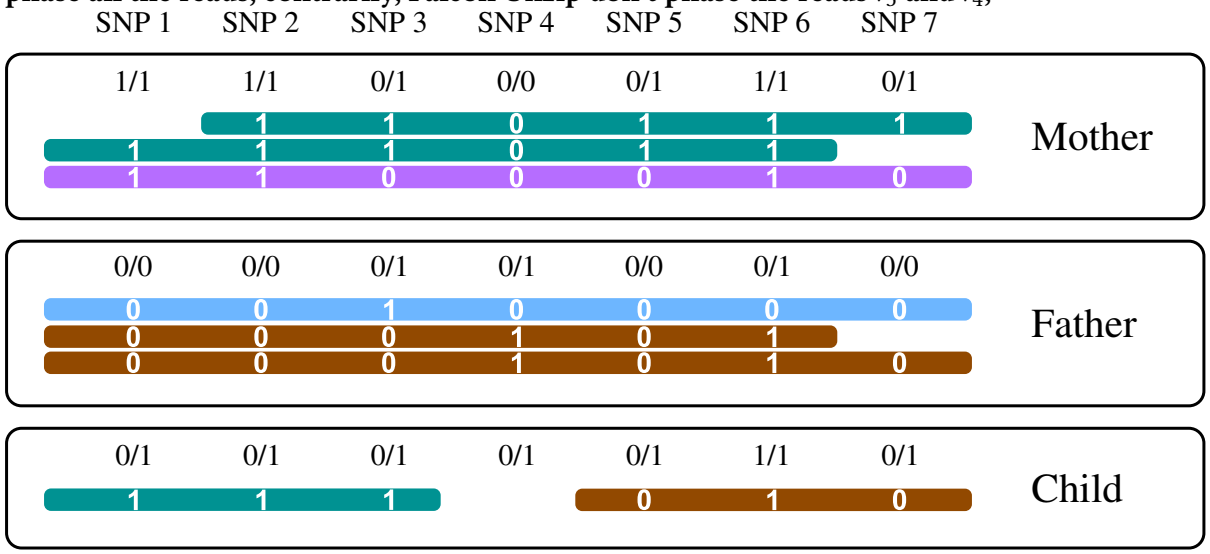


Figure 1.9: Seven SNP loci covered by reads (horizontal bars) in three individuals. Unphased genotypes are indicated by labels 0/0, 0/1 and 1/1. The alleles that a read supports are printed in white

between individuals within a pedigree. At the simplest level of a family trio (both parents and one child), very simple rules indicate which alleles in the child were inherited from each parent, thus largely separating the two haplotypes in the child. Nevertheless, genetic analysis cannot phase positions in which all family members are heterozygous. Furthermore, it is not always feasible to recruit the required participants for family-based studies. In the absence of a family context, molecular haplotyping is an excellent choice because it does not require DNA samples from other family members. The sequencing based haplotyping largely supplant the need for genetic analysis. Haploscribe (Roach et al., 2011b) is a suite of software scripts that phase whole-genome data across entire chromosomes by genetic analysis. Haploscribe implements a parsimony approach to generate meiosis-indicator (inheritance state) vectors and uses a hidden Markov model (HMM) to deduce haplotypes spanning entire chromosomes. To our knowledge, we are not aware of any method that uses both sequencing data and genetic inheritance

principles in an integrative fashion to perform phasing.

Open Problem 1.4. Combining both principles of genetic inheritance and sequencing reads into one framework is not yet solved. Furthermore, the parameterized algorithm to solve this integrative framework is an important question.

Example 1.5. To illustrate the motivation to combine genetic and read-based haplotyping, the corresponding MEC instance for a single genome is shown in Figure 1.9. There are seven SNP positions covered by reads in three related individuals. It 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. By observing that all seven child genotypes are compatible with the combination of brown and green haplotypes from the parents, however, these phases can be easily inferred. This example demonstrates that jointly using pedigree information, genotypes and sequencing reads is very powerful for establishing phase information.

1.4 Outline of our contributions

In the above section, we highlighted four "open problems" in the diploid genome assembly of NGS data:

1.4.1 Open Problems

1. Approximation status of Gapless-MEC.
2. Integrative framework for haplotyping using different types of datasets.
3. Denovo diploid genome assembly.
4. Phasing pedigree of genomes.

1.4.2 Issues we address

To address those problems, we will present new algorithmic approaches.

1. In the first chapter, we present dynamic programming based algorithm to prove the near-polynomial approximation status of Gapless-MEC.
2. In the second chapter, we present a parameterized algorithm to solve MEC instances integatively from different datasets.
3. In the third chapter, we introduce new way to represent the assembly graph and futher, finding long read paths in the graph based on different types of datasets, helps in better phasing.
4. In the forth chapter, we present a integrative framework to solve sequencing-based and genetic haplotyping, helps to generate complete and accurate haplotypes.

Chapter **2**

Biological and Algorithmic background

In the first part of this chapter we provide explanation of some of the biological concepts we introduced in the earlier chapter. This explanation is required for understanding the material presented later in the thesis. In the second part we provide types of algorithms used in computer science to solve biological problems that can be formulated as optimization problem.

2.1 Biological Background

2.1.1 DNA structure

As we learnt in the previous section, the genetic information of different organisms is encoded in DNA. The DNA molecule is a chain on which many bases are ordered in a linear sequence, the bases — A, T, G, and C — as the letters of a genetic alphabet. The DNA sequence length can be short, for example, 12 million nucleotides long for yeast genomes, or long in the order of approximately 3 billion nucleotides for humans. One can think of DNA as a "genetic database" for organisms.

DNA stands for deoxyribonucleic acid. By breaking the name itself suggests the structure of the molecule, which consists of three components: 1) a sugar molecule, 2) a phosphate group, and 3) a nitrogenous base. The nitrogenous bases are what make DNA variable. There are 4 different types of bases in DNA:adenine, guanine, thymine, and cytosine. Biologists commonly abbreviate these bases as the letters A, G, T, and C, respectively. Each one of the bases is chemically distinguishable from the others, it is the variability of these bases that constitutes the genetic code.

Futhermore, a double helix of DNA is composed of two spiraling, complementary strands of DNA. Each strand is composed of a sugar and phosphate backbone with varying nitrogenous bases sticking in towards the center. The two strands are joined together at the center by pairing bases lined up with one another. DNA is often described structurally as a *twisting ladder*. In this ladder, the "rungs" are the pairs of bases linked together, and the "sides" are the two separate sugar and phosphate backbones. The double helix is important because it preserves all of the information-carrying features of a single DNA strand while at the same time introducing elements that make it easier for living cells to make copies of their DNA. Because every base pair in the double helix must match its pairing partner (A with T, C with G), we can easily determine the sequence of an unknown strand of DNA if its matching strand is known. For example, if one strand of a double helix has the nucleotide sequence GATTCGTACG, then its complementary strand will be CTAAGCATGC.

2.1.1.1 Chromosomes

DNA is divided into bundles known as chromosomes. Chromosomes have several important features. First of all, the DNA packs so tightly that one can see it under a simple light microscope. Secondly, recall that because the cell is getting ready to divide in two, the DNA of a visible chromosome has already been duplicated, so that each successor cell will have its own copy. This means that, on close inspection, a cell that is ready to divide will have four strands of DNA, two helices of two strands each.

Each of these double strands of DNA condenses into a single rod called a sister chromatid. The two chromatids are therefore exact replicas of one another, and the center of each is joined together prior to the division of the cell. As a result, most chromosomes take on the appearance of the letter X.

The human genome is composed of 23 kinds of chromosomes. However, because humans conceive through sexual reproduction, every child receives two sets of 23 chromosomes – one from mother and the other from father. As a result, every individual has 23 pairs of chromosomes, for a total of 46. Of these 23 pairs, one pair is responsible for determining sex. The chromosomes in this pair are therefore called sex chromosomes. The chromosomes in the remaining 22 pairs are called autosomes. The two chromosomes in a pair of autosomes are called homologues, or a "homologous pair," meaning that they contain corresponding sequences of DNA given in Figure 2.1. These two chromosomes come from separate parents. The homologous chromosomes contain DNA sequences that are similar, but they are not identical copies of each other.

2.1.2 Diploid nature of organisms

The diploid nature of organisms occur due to the process of mitosis.

Concept of Mitosis Figure 2.2 is a simplified diagram illustrating the overall process of mitosis and the detailed phases are in given in Figure 2.3. In the first step, called interphase, the DNA strand of a chromosome is copied (the DNA strand is replicated) and this copied strand is attached to the original strand at a spot called the centromere. This new structure is called a bivalent chromosome. A bivalent chromosome consists of two sister chromatids (DNA strands that are replicas of each other). When a chromosome exists as just one chromatid, just one DNA strand and its associated proteins, it is called a monovalent chromosome. The second and third steps of mitosis organize the newly created bivalent chromosomes so that they they can be split in an orderly fashion. In the second step, prophase, the bivalent chromosomes condense into tight packages. In the third step of mitosis, called metaphase, each chromosome lines up in a single file line at the center of the cell. In the fourth step, anaphase, the mitotic spindles pry each chromatid apart from its copy, and drag them to the opposite side of the cell. Four bivalent chromosomes become two groups of 4 monovalent chromosomes.

Concept of Meiosis The purpose of meiosis is to make haploid gametes. In order to explain the difference between mitosis and meiosis quickly and easily, consider the following analogy: You own a restaurant, and you keep 46 cookbooks on hand, to store all the recipes you need to make the food you sell. If you opened a new restaurant that you wanted to make the same food as the one that already exists, what would you do? Copy all 46 cookbooks, and take them to the new restaurant. That's like what happens in mitosis. Consider that the cookbooks are chromosomes, each containing lots of recipes that cells use to make "dishes," called proteins. When cell division occurs, each cell wants to ensure that each new cell can make the same proteins as the original. So each of the chromosomes are copied and evenly distributed to both new cells—both cells get a copy of each "cookbook." Meiosis is different. Whereas as mitosis makes a new cell with the same number of chromosomes, meiosis is a reductive type of cell division: it results in cells with fewer chromosomes.

Figure 2.4 is a simplified diagram illustrating the overall process and products of meiosis.

Meiosis is split into two separate parts, called meiosis I and meiosis II. Meiosis I starts with the copying of chromosomes and their condensation into compact forms (just like mitosis). The metaphase of meiosis I is different, though: Instead of lining up in single file, the bivalent chromosomes line up two-by-two. These groups, called homologous chromosomes, are what separate during the anaphase of meiosis I (compare this to the anaphase of mitosis, where chromatids separate).

If we look at the anaphase of meiosis I in nematodes (diploid number 4), the result is two groups of two bivalent chromosomes, rather than two groups of four monovalent ones. This difference in chromosome number in the post-anaphase groupings is really the only big difference between meiosis I

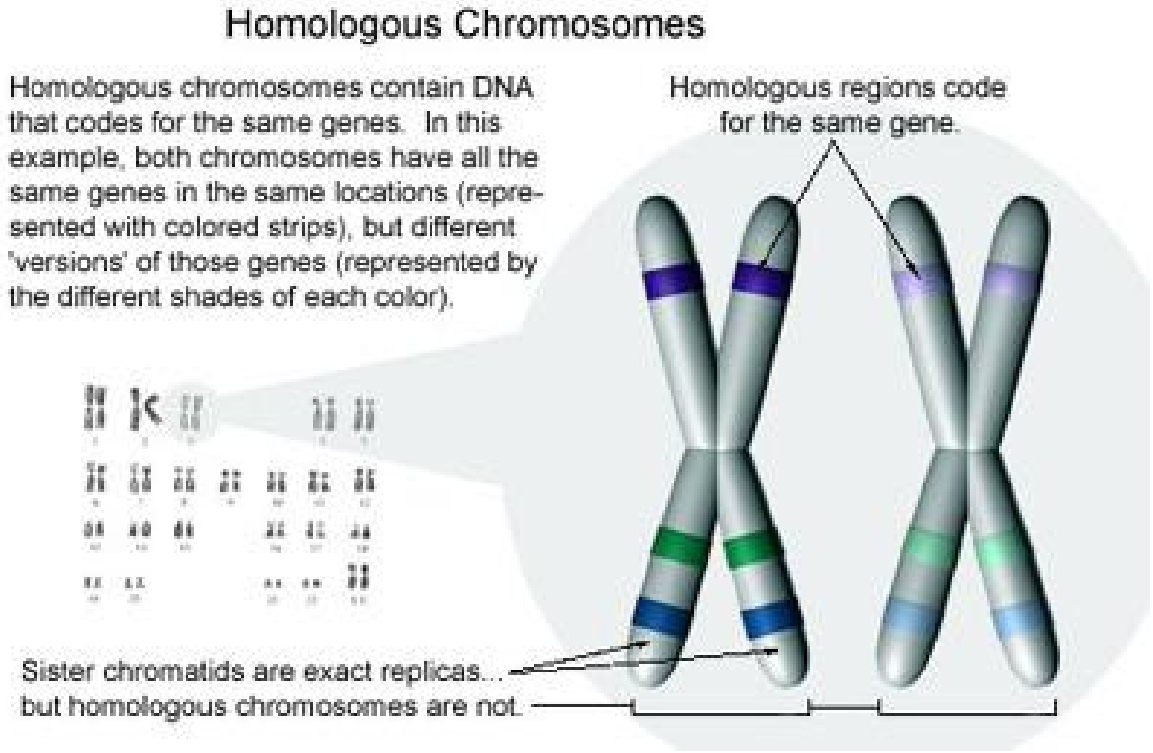


Figure 2.1: Homologous chromosomes, sister chromatids.

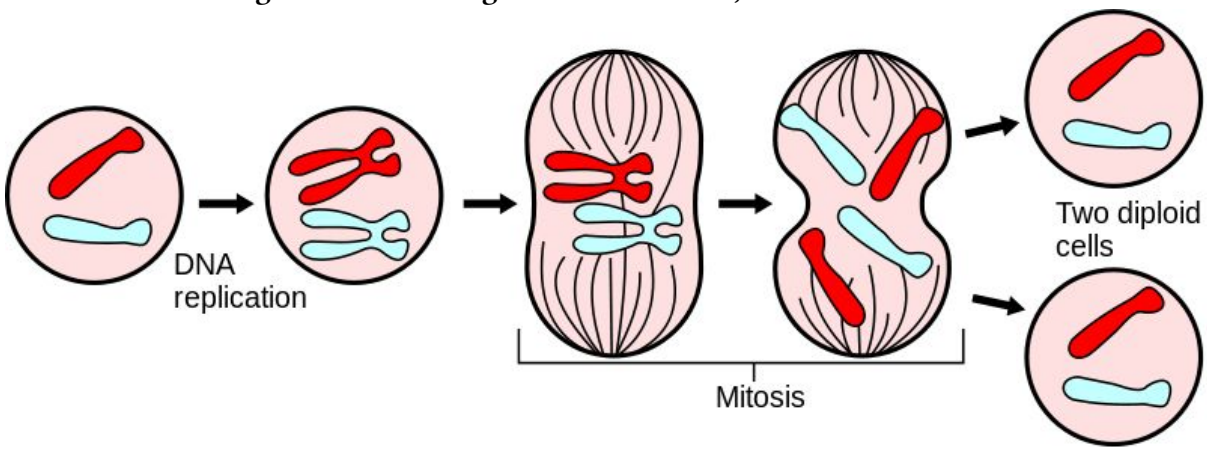


Figure 2.2: Overview of Mitosis.

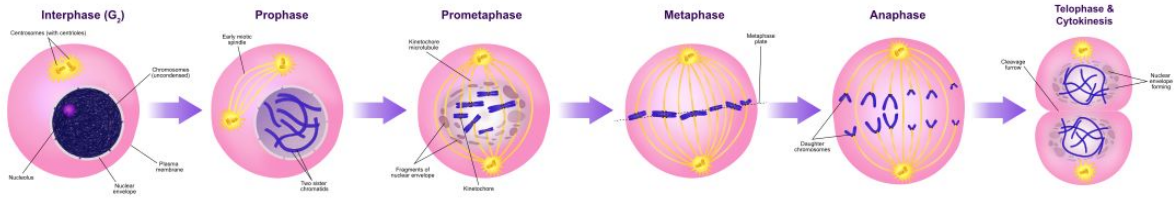


Figure 2.3: Phases of cell cycle and mitosis

and mitosis. Membranes form around the two groups, during telophase, and then the cell splits down the middle creating two non-clones. Each clone has half the number of chromosomes as the initial cell.

Meiosis II applies the process of mitosis to the two cells created by meiosis I. Since the chromosomes already exist in the bivalent form, interphase is skipped. The result is four cells, called gametes, each

with two monovalent chromosomes. Meiosis sets the stage for Mendelian genetics. Students need to know that most of the genetics action occurs in the first meiotic division:

1. homologous chromosomes pair up and align end-to-end (synapsis) in prophase I
2. crossing over occurs between homologous chromosomes in prophase I, before chromosomes line up at the metaphase plate
3. homologous chromosomes separate to daughter cells (sister chromatids do not separate) in the first division, creating haploid (1N) cells
4. the separation of each pair of homologous chromosomes occurs independently, so all possible combinations of maternal and paternal chromosomes are possible in the two daughter cells – this is the basis of Mendel's Law of Independent Assortment
5. the first division is when daughter cells become functionally or genetically haploid

Let's discuss the last point more in detail. Consider the X and Y chromosomes. They pair in prophase I, and then separate in the first division. The daughter cells of the first meiotic division have either an X or a Y; they don't have both. Each cell now has only one sex chromosome, like a haploid cell.

One way of thinking about ploidy is the number of possible alleles for each gene a cell can have. Right after meiosis I, the homologous chromosomes have separated into different cells. Each homolog carries one copy of the gene, and each gene could be a different allele, but these two homologs are now in two different cells. Though it looks like there are two of each chromosome in each cell, these are duplicated chromosomes; ie, it is one chromosome which has been copied, so there is only one possible allele in the cell (just two copies of it).

The second meiotic division is where sister (duplicated) chromatids separate. It resembles mitosis of a haploid cell. At the start of the second division, each cell contains 1N chromosomes, each consisting of a pair of sister chromatids joined at the centromere.

Genetic recombination Recombination occurs during meiosis and is a process that breaks and recombines pieces of DNA to produce new combinations of genes. Recombination scrambles pieces of maternal and paternal genes, which ensures that genes assort independently from one another. It is important to note that there is an exception to the law of independent assortment for genes that are located very close to one another on the same chromosome because of genetic linkage.

During fertilisation, 1 gamete from each parent combines to form a zygote. Because of recombination and independent assortment in meiosis, each gamete contains a different set of DNA. This produces a unique combination of genes in the resulting zygote.

Recombination or crossing over occurs during prophase I. Homologous chromosomes – 1 inherited from each parent – pair along their lengths, gene by gene. Breaks occur along the chromosomes, and they rejoin, trading some of their genes. The chromosomes now have genes in a unique combination.

Independent assortment is the process where the chromosomes move randomly to separate poles during meiosis. A gamete will end up with 23 chromosomes after meiosis, but independent assortment means that each gamete will have 1 of many different combinations of chromosomes.

This reshuffling of genes into unique combinations increases the genetic variation in a population and explains the variation we see between siblings with the same parents.

2.1.3 The Era of NGS

Since the completion of the human genome project in 2003, extraordinary progress has been made in genome sequencing technologies, which has led to a decreased cost per megabase and an increase in the number and diversity of sequenced genomes. An astonishing complexity of genome architecture has been revealed, bringing these sequencing technologies to even greater advancements. Some approaches maximize the number of bases sequenced in the least amount of time, generating a wealth of data that can be used to understand increasingly complex phenotypes. Alternatively, other approaches now aim to sequence longer contiguous pieces of DNA, which are essential for resolving structurally complex regions. These and other strategies are providing researchers and clinicians a variety of tools to probe

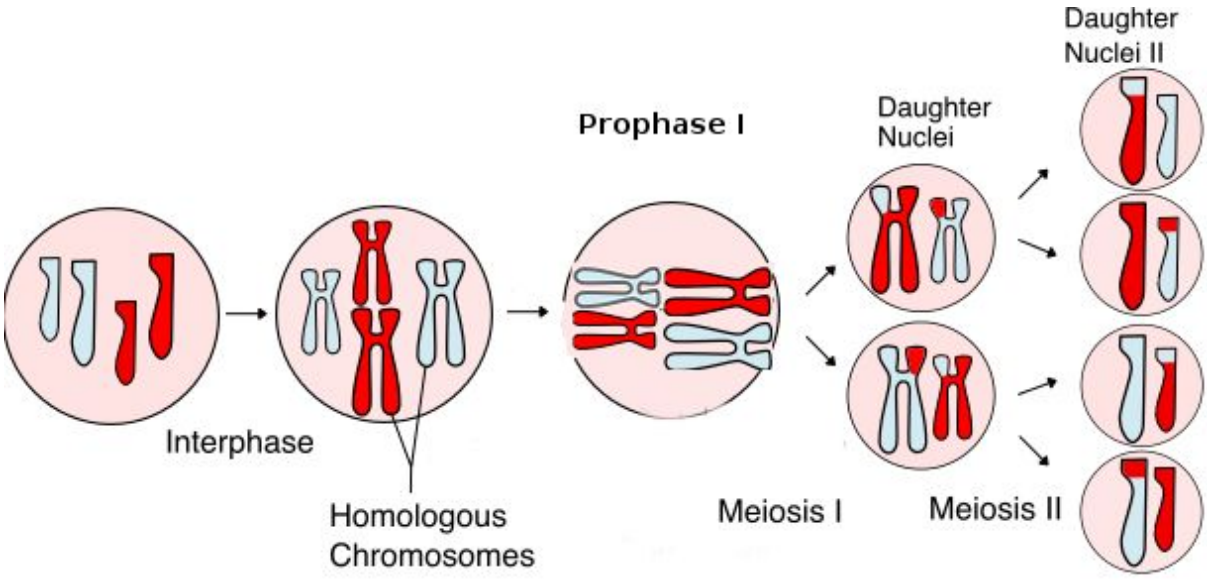


Figure 2.4: Overview of Meisosis.

genomes in greater depth, leading to an enhanced understanding of how genome sequence variants underlie phenotype and disease.

The different advancements come with their own limitations. As new technologies emerge, existing problems are exacerbated or new problems arise. NGS platforms provide vast quantities of data, but with associated error rates (0.1-15%) are higher and the read-lengths generally shorter (35-700) bps for short-read approaches, than those of traditional *Sanger* sequencing platforms, require careful examination of the results, particularly for variant discovery and clinical applications. Although long-read sequencing overcome the length limitation of other NGS platforms, it remains considerably more expensive and has lower throughput than other platforms, limiting the wide-spread adoption of this technology in favour of less-expensive approaches.

2.1.3.1 Short Read NGS

Illumina's suite of instruments for short-read sequencing range from small, low-throughput benchtop units to large untra-high through instruments dedicated to population-level whole genome sequencing. dNTP identification is achieved through total internal reflection flourescence microscopy using two or four laser channels. In most Illumina platforms, each dNTP is bound to a single flourescence that is specific to that base type and requires four different imaging channels, whereas NextSeq and Mini-Seq systsme use a two-fluorophone system. The first NGS instrument developed was the 454 pyrosequencing (Margulies et al., 2005) device. This SNA system distributes template-bound beads into a PicoTiterPlate along with beads containing an enzyme cocktail. As a dNTP is incorporated into a strand, an enzymatic cascade occurs, resulting in a bioluminescence signal. Each burst of light, detected by a charge-coupled device (CCD) camera, can be attributed to the incorporation of one or more identical dNTPs at a particular bead (Figure 2.5).

The Ion Torrent was the first NGS platform without optical sensing (Rothberg et al., 2011). Rather than using an enzymatic cascade to generate a signal, the Ion Torrent platform detects the H + ions that are released as each dNTP is incorporated. The resulting change in pH is detected by an integrated complementary metal-oxide- semiconductor (CMOS) and an ion-sensitive field-effect transistor (ISFET) (Figure 2.6). The pH change detected by the sensor is imperfectly proportional to the number of nucleotides detected, allowing for limited accuracy in measuring homopolymer lengths.

The Illumina CRT system (Figure 2.7) accounts for the largest market share for sequencing instruments compared to other platforms. Illumina's suite of instruments for short-read sequencing range from small, low-throughput benchtop units to large ultra-highthroughput instruments dedicated to population-level whole-genome sequencing (WGS). dNTP identification is achieved through total inter-

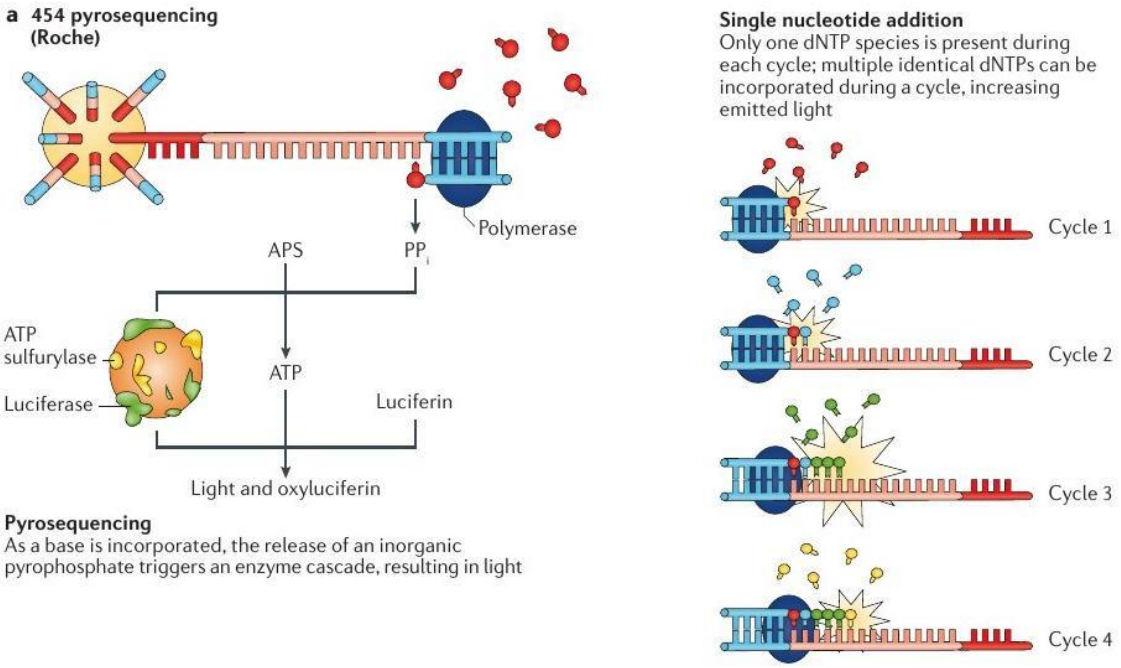


Figure 2.5: Overview of 454 pyrosequencing.

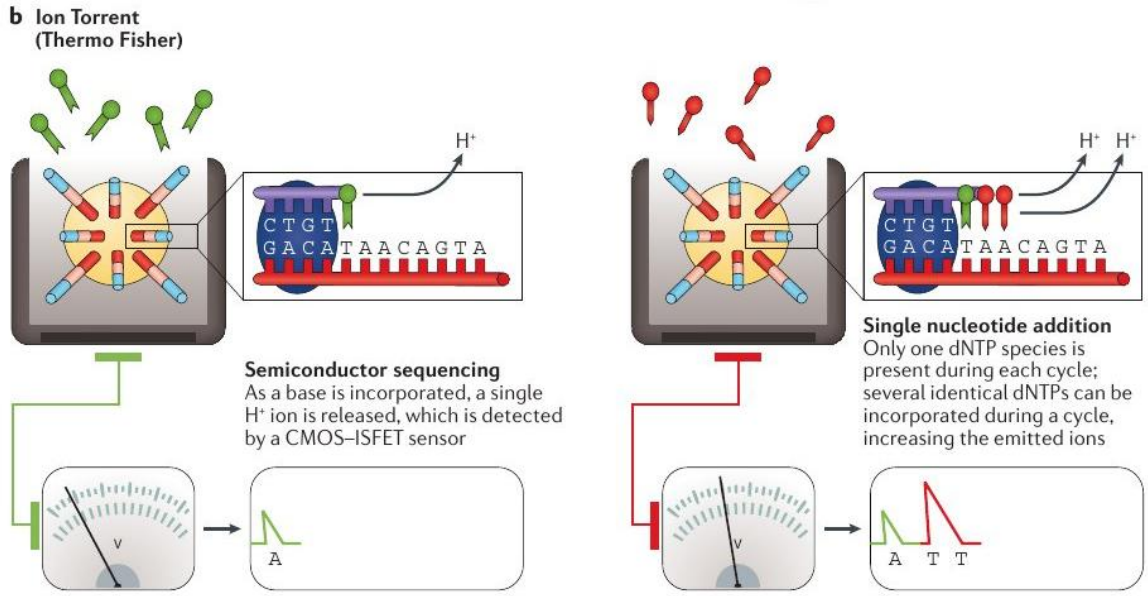


Figure 2.6: Overview of Ion Torrent sequencing.

nal reflection fluorescence (TIRF) microscopy using either two or four laser channels. In most Illumina platforms, each dNTP is bound to a single fluorophore that is specific to that base type and requires four different imaging channels, whereas the NextSeq and Mini-Seq systems use a two-fluorophore system.

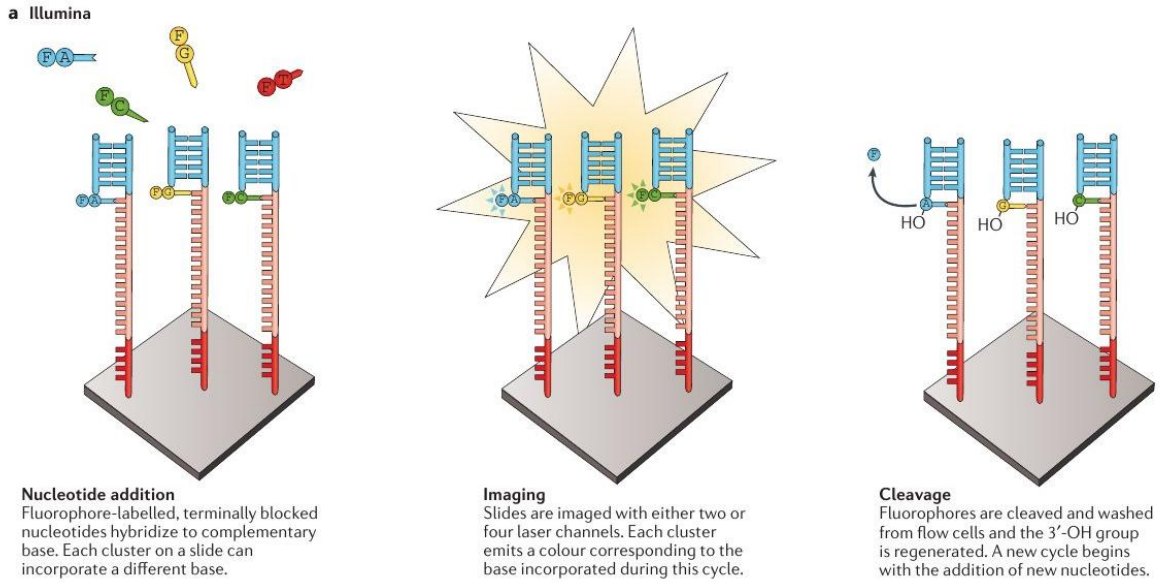


Figure 2.7: Overview of Illumina sequencing.

Comparison of short-read platforms. Individual short-read sequencing platforms vary with respect to throughput, cost, error profile and read structure (**[TODO: TABLE 1]**). Despite the existence of several NGS technology providers, NGS research is increasingly being conducted within the Illumina suite of instruments. Although this implies high confidence in their data, it also raises concerns about systemic biases derived from using a single sequencing approach. As a consequence, new approaches are being developed and researchers increas-ingly have the choice to integrate multiple sequencing methods with complementary strengths. The SBL technique used by both the SOLiD and Complete Genomics systems affords these technologies a very high accuracy (99.99%)(Liu et al., 2012; Drmanac et al., 2010), as each base is probed multiple times. Although accurate, both platforms also show evidence of a trade-off between sensitivity and specificity, such that true variants are missed while few false variants are called. There is also evidence that the platforms share some under- representation of AT-rich regions, and the SOLiD platform displays some substitution errors and some GC-rich under-representation. Perhaps the feature most limiting to the widespread adoption of these technologies is the very short read lengths. Although both platforms can generate single-end and paired-end sequencing reads, the maximum read length is just 75 bp for SOLiD and 28–100 bp for Complete Genomics, limiting their use for genome assembly and structural variant detection applications. Unfortunately, owing to these limitations, along with runtimes on the order of several days, the SOLiD system has been relegated to a small niche within the industry. Illumina dominates the short-read sequencing industry owing, in part, to its maturity as a technology, a high level of cross-platform compatibility and its wide range of platforms. The suite of instruments available ranges from the low-throughput MiniSeq to the ultra-high-throughput HiSeq X, which is capable of sequencing 1,800 human genomes to 30× coverage per year. Further diversification is derived from the many options available for runtime, read structure and read length (up to 300 bp). As the Illumina platform relies on a CRT approach, it is much less susceptible to the homopolymer errors observed in SNA platforms. Although it has an overall accuracy rate of >99.5%,the platform does display some under-representation in AT-rich and GC-rich regions, as well as a tendency towards substitution errors. In 2008, Bentley et al. (2008) reported a very high concordance rate between human

single-nucleotide polymorphisms (SNPs) identified with Illumina and SNPs identified from genotyping microarrays. However, this high sensitivity came with a false-positive rate of around 2.5%, leading this and other groups to consider using Sanger sequencing to resequence the called SNPs in order to distinguish between true SNPs and false positives. With all of the possible options available, the Illumina suite allows for a wide range of applications: genome sequencing through WGS or exome sequencing; epi-genomics applications, such as ChIP–seq (chromatin immunoprecipitation followed by sequencing), ATAC –seq (assay for transposase- accessible chromatin using sequencing) or DNA methylation sequencing (methyl-seq); and transcriptomics applications through RNA sequencing (RNA-seq), to name a few. HiSeq X is currently the highest throughput instrument available; however, as a consequence of its optimization, it is limited to just a few applications, such as WGS and whole-genome bisulfite sequencing. HiSeq X is further limited as an all-purpose instrument owing to a required initial purchase of five or ten instruments (additional single instruments can be purchased after the initial commitment), placing this system out of reach of most facilities.

Both the 454 and the Ion Torrent systems offer superior read lengths compared to other short-read sequencers with reads up to an average of 700 bp and 400 bp, respectively, providing some advantages for applications that focus on repetitive or complex DNA. However, as both of these platforms rely on SNA, they share many of the same drawbacks. Insertion and deletion (indel) errors dominate, although the overall error rate is on par with other NGS platforms in non-homopolymer regions. Homopolymer regions are problematic for these platforms, which lack single-base accuracy in measuring homopolymers larger than 6–8 bp (Forgetta et al., 2013; Loman et al., 2012). Unfortunately, whereas the Ion Torrent platform has kept pace with the rapidly evolving NGS field, the 454 platform has been unable to complete with other platforms in terms of yield or cost.

2.1.3.2 Long-read sequencing

It has become apparent that genomes are highly complex with many long repetitive elements, copy number alterations and structural variations that are relevant to evolution, adaptation and disease (McCarroll and Altshuler, 2007; Stankiewicz and Lupski, 2010). However, many of these complex elements are so long that short-read paired-end technologies are insufficient to resolve them. Long-read sequencing delivers reads in excess of several kilobases, allowing for the resolution of these large structural features. Such long reads can span complex or repetitive regions with a single continuous read, thus eliminating ambiguity in the positions or size of genomic elements. Long reads can also be useful for transcriptomic research, as they are capable of spanning entire mRNA transcripts, allowing researchers to identify the precise connectivity of exons and discern gene isoforms.

Currently, there are two main types of long-read technologies: single-molecule real-time sequencing approaches and synthetic approaches that rely on existing short- read technologies to construct long reads in silico. The single-molecule approaches differ from short-read approaches in that they do not rely on a clonal population of amplified DNA fragments to generate detectable signal, nor do they require chemical cycling for each dNTP added. Alternatively, the synthetic approaches do not generate actual long-reads; rather, they are an approach to library preparation that leverages barcodes to allow computational assembly of a larger fragment.

Single-molecule long-read sequencing (PacBio and ONT) Currently, the most widely used long-read platform is the single-molecule real-time (SMRT) sequencing approach used by Pacific Biosciences (PacBio) (Eid et al., 2009) (Figure 2.8a). The instrument uses a specialized flow cell with many thousands of individual picolitre wells with transparent bottoms — zero-mode waveguides (ZMW) (Levene et al., 2003). Whereas short-read SBS technologies bind the DNA and allow the polymerase to travel along the DNA template, PacBio fixes the polymerase to the bottom of the well and allows the DNA strand to progress through the ZMW. By having a constant location of incorporation owing to the stationary enzyme, the system can focus on a single molecule. dNTP incorporation on each single- molecule template per well is continuously visualized with a laser and camera system that records the colour and duration of emitted light as the labelled nucleotide momentarily pauses during incorporation at

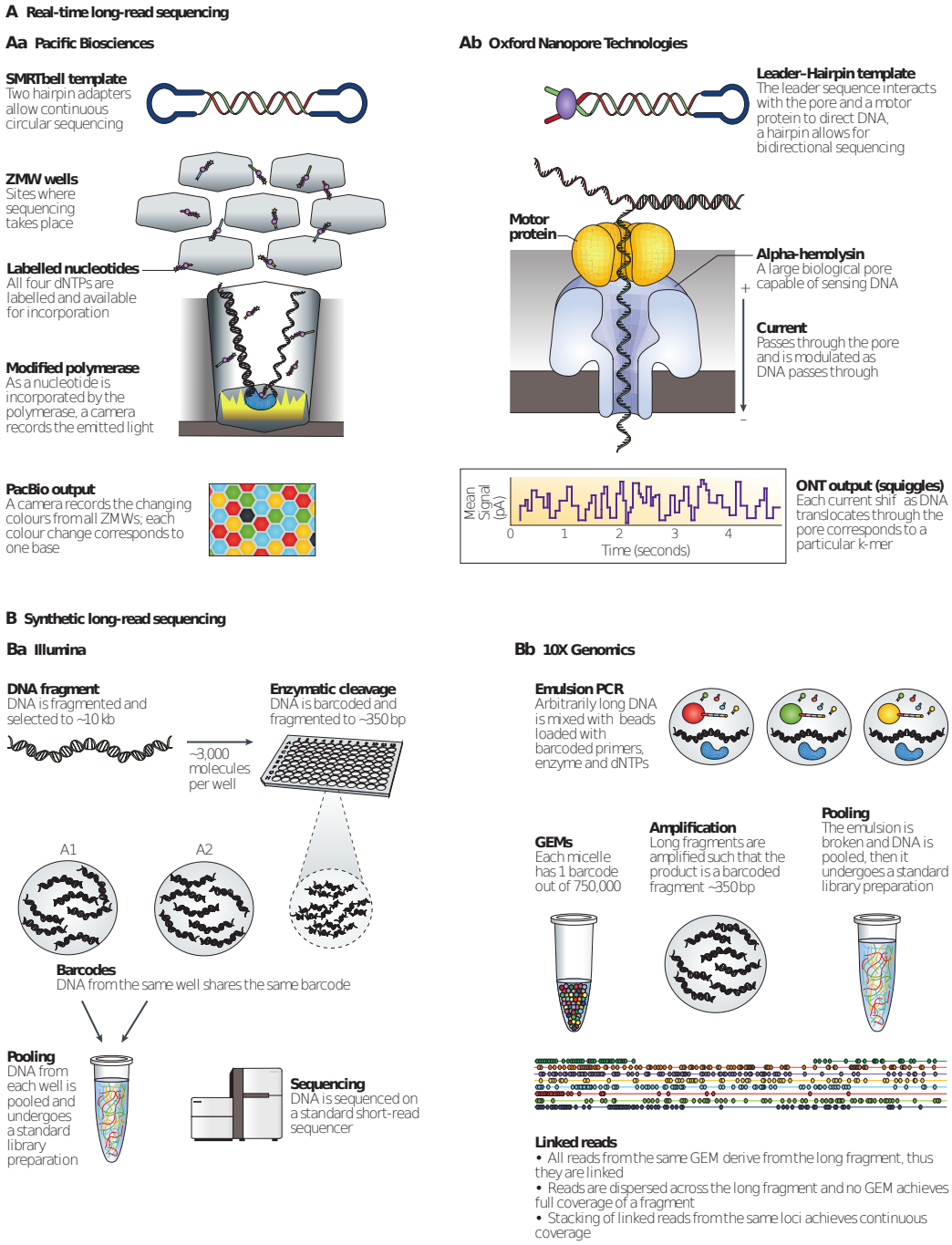


Figure 2.8: Overview of long-read sequencing and synthetic long reads.

the bottom of the ZMW. The polymerase cleaves the dNTP-bound fluorophore during incorporation, allowing it to diffuse away from the sensor area before the next labelled dNTP is incorporated. The SMRT platform also uses a unique circular template that allows each template to be sequenced multiple times as the polymerase repeatedly traverses the circular molecule. Although it is difficult for DNA templates longer than 3 kb to be sequenced multiple times, shorter DNA templates can be sequenced

many times as a function of template length. These multiple passes are used to generate a consensus read of insert , known as a circular consensus sequence (CCS).

In 2014, the first consumer prototype of a nanopore sequencer — the MinION from Oxford Nanopore Technologies (ONT) — became available. Unlike other platforms, nanopore sequencers do not monitor incorporations or hybridizations of nucleotides guided by a template DNA strand. Whereas other platforms use a secondary signal, light, colour or pH, nanopore sequencers directly detect the DNA composition of a native ssDNA molecule. To carry out sequencing, DNA is passed through a protein pore as current is passed through the pore (Clarke et al., 2009)(Figure 2.8b). As the DNA translocates through the action of a secondary motor protein, a voltage blockade occurs that modulates the current passing through the pore. The temporal tracing of these charges is called squiggle space, and shifts in voltage are characteristic of the particular DNA sequence in the pore, which can then be interpreted as a k-mer. Rather than having 1–4 possible signals, the instrument has more than 1,000 — one for each possible k-mer, especially when modified bases present on native DNA are taken into account. The current MK1 MinION flow cell structure is composed of an application-specific integrated circuit (ASIC) chip with 512 individual channels that are capable of sequencing at 70 bp per second, with an expected increase to 500 bp per second in 2016. The upcoming PromethION instrument is intended to be an ultra-high-throughput platform reported to include 48 individual flow cells, each with 3,000 pores running at 500 bp per second. This works out to 2–4 Tb for a 2-day run on a fully loaded device, placing this device in potential competition with Illumina's HiSeq X. Similar to the circular template used by PacBio, the ONT MinION uses a leader-hairpin library structure. This allows the forward DNA strand to pass through the pore, followed by a hairpin that links the two strands, and finally the reverse strand. This generates 1D and 2D reads in which both '1D' strands can be aligned to create a consensus sequence '2D' read.

Synthetic long reads. Unlike true sequencing platforms, synthetic long-read technology relies on a system of barcoding to associate fragments that are sequenced on existing short-read sequencers. These approaches partition large DNA fragments into either microtitre wells or an emulsion such that very few molecules exist in each partition. Within each partition the template fragments are sheared and barcoded. This approach allows for sequencing on existing short-read instrumentation, after which data are split by barcode and reassembled with the knowledge that fragments sharing barcodes are derived from the same original large fragment (McCoy et al., 2014). Similar to an earlier technology, BAC-by-BAC sequencing, synthetic barcoded reads provide an association among small fragments derived from a larger one. By segregating the fragments, repetitive or complicated regions can be isolated, allowing each to be assembled locally. This prevents unresolvable branch points in the assemblies, which lead to breaks (gaps) and shorter assembled contiguous sequences.

There are currently two systems available for generating synthetic long-reads: the Illumina synthetic long-read sequencing platform (Figure 2.8c) and the 10X Genomics emulsion-based system (Figure 2.8d). The Illumina system (formerly Moleculo) partitions DNA into a microtitre plate and does not require specialized instrumentation. However, the 10X Genomics instruments (GemCode and Chromium) use emulsion to partition DNA and require the use of a microfluidic instrument to perform pre-sequencing reactions. With as little as 1 ng of starting material, the 10X Genomics instruments can partition arbitrarily large DNA fragments, up to 100 kb, into micelles called 'GEMs', which typically contain $\leq 0.3\times$ copies of the genome and one unique barcode. Within each GEM, a gel bead dissolves and smaller fragments of DNA are amplified from the original large fragments, each with a barcode identifying the source GEM. After sequencing, the reads are aligned and linked together to form a series of anchored fragments across the span of the original fragment. Unlike the Illumina system, this approach does not attempt gapless, end-to-end coverage of a single DNA fragment. Instead it relies on linked reads, in which dispersed, small fragments that are derived from a single long molecule share a communal barcode. Although these fragments leave segments of the original large molecule without any coverage, the gaps are overcome by ensuring that there are many long fragments from the same genomic region in the initial preparation, thus generating a read cloud wherein linked reads from each long fragment

can be stacked, combining their individual coverage into an overall map(Figure 2.8d).

Comparison of single-molecule and synthetic long-read sequencing. There is growing interest in the field of long-read sequencing, and each system has its own advantages and drawbacks (**[TODO: TABLE 1]**). Currently, the most widely used instrument in long-read sequencing is the PacBio RS II instrument. This device is capable of generating single polymerase reads in excess of 50 kb with average read lengths of 10–15 kb for a long-insert library. Such properties are ideal for de novo genome assembly applications, for revealing complex long-range genomic structures and for full-length transcript sequencing. There are, however, several notable limitations. The single- pass error rate for long reads is as high as 15% with indel errors dominating, raising concerns about the utility of the instrument. Fortunately, these errors are randomly distributed within each read and hence sufficiently high coverage can overcome the high error rate. The use of a circular template by PacBio also provides a level of error correction. The more frequently a single molecule is sequenced, the higher the resulting accuracy — up to 99.999% for insert sequences derived from at least 10 subreads.This high accuracy rivals that of Sanger sequencing, leading researchers to speculate that this technology can be used in a manner analogous to Sanger-based SNP validation. The runtimes and throughput of this instrument can be tuned by controlling the length of time for which the sensor monitors the ZMW; longer templates require longer times. For example, a 1 kb library that is run for 1 hour will generate around 7,500 bases of sequence per molecule, with an average of 8 passes, whereas a 4-hour run will generate around 30,000 bases per molecule and 30 passes. Conversely, a 10 kb library requires a 4-hour run to generate 30,000 bases with 3 passes. The limited throughput and high costs of PacBio RS II (around 1,000 per Gb), in addition to the need for high coverage, place this instrument out of reach of many small laboratories. However, in an attempt to ameliorate these concerns, PacBio has launched the Sequel System, which reportedly has a throughput 7× that of the RS II, thus halving the cost of sequencing a human genome at 30× coverage.

The ONT MinION is a small (3 cm× 10 cm for the MK1) USB-based device that runs off a personal computer, giving it the smallest footprint of any current sequencing platform. This affords the MinION superior portability, highlighting its utility for rapid clinical responses and hard-to-reach field locations. Although substantial adjunct equipment is still required for library preparation (for instance, a thermocycler), improvements in library preparation and equipment optimization could conceivably reduce the space required for a fully functional sequencing system to the size of a single bag of luggage. Unlike other platforms, the MinION has few constraints on the size of the fragments to be sequenced. In theory, a DNA molecule of any size can be sequenced on the device, but in practice there are some limitations when dealing with ultra-long fragments. As a consequence of the unique nature of the ONT technology, in which there are more than 1,000 distinct signals, ONT MinION has a large error rate — up to 30% for a 1D read — and is dominated by indel errors. Effective homopolymer sequencing also remains a challenge for ONT MinION. When a homopolymer exceeds the k-mer length, it can be difficult to identify when one k-mer leaves the pore and another k-mer enters. Modified bases also pose a challenge to the device, as a modified base will alter the typical voltage shift for a given k-mer. Fortunately, recent improvements in the chemistry and the base calling algorithms are improving accuracy.

The Illumina synthetic long-read approaches are a direct response to the costs, error rates and throughput of true long-read sequencers. Relying on the existing Illumina infrastructure affords researchers the ability to simply purchase a kit for long-read sequencing. Accordingly, the throughput and error profile are identical to those of current Illumina devices. However, as a consequence of how the DNA is partitioned, the system requires more coverage than is required for a typical short-read project, thus increasing the costs associated with this technology relative to other Illumina applications.

Like the Illumina synthetic long-read platform, the 10X Genomics emulsion-based platform relies on an existing short-read infrastructure to provide the sequencing. The microfluidic instrument is a one-time additional equipment cost, and the emulsion approach used allows for as little as 1 ng of starting material, which can be beneficial for situations in which the DNA is precious, such as biopsy samples. Currently, data output from the GemCode instrument is partially limited by the number of

barcodes used and the somewhat inefficient DNA partitioning. Inefficient partitioning can lead to a surplus of DNA fragments within a droplet, thus complicating sequence deconvolution, which is further exacerbated by the limited number of barcodes. Both of these conditions lead to ambiguity regarding the positional relationship between reads sharing the same barcode, making analysis more difficult.

2.1.3.3 Strand-Seq sequencing

Strandseq (Falconer et al., 2012; **?**) is a single cell sequencing technique that identifies the original parental DNA template strands in daughter cells following cell division. The method uses bromodeoxyuridine (BrdU) incorporation in the nascent strand during DNA replication followed by selective degradation of the nascent strand to isolate the template strand for construction of directional sequencing libraries.

Strand-seq is a single-cell template strand sequencing technology that generates directional genomic libraries that, when aligned to the reference genome, permits a clear distinction between the individual homologs of a chromosome. Homolog resolution allows diverse types of variants to be located in a single cell that would otherwise be very challenging to detect using conventional approaches. This includes inversions, translocations, copy-number changes, aneuploidy events and complex structural variants, with haplotype awareness. By pooling data from several dozen Strand-seq cells, each homologous chromosome can be uniquely characterized and analyzed.

In Strand-seq (Fig. 2.9), parental DNA template strands inherited by daughter cells are sequenced following construction of a modified Illumina genomic library20. The method takes advantage of the directionality of single-stranded DNA molecules, which are distinguished as either Crick (C; forward, or plus, strand of the reference assembly) or Watson (W; reverse, or minus, strand) based on their 5–3 orientation (Fig. 2.9a). Daughter cells, generated following one mitotic division of a parental cell in the presence of BrdU, are isolated and deposited as single cells (or nuclei) into single wells of a 96-well plate (Fig. 2.9b). Forgoing any whole-genome preamplification step, the genomic DNA is fragmented using micrococcal nuclease (MNase) to generate mononucleosomal fragments of 150 bp in length. The MNase-digested chromatin is then blunt-ended and 5-phosphorylated by an end repair reaction, and a single adenine is attached to the 3 end of the fragment by an A-tailing reaction. This adenine serves as a substrate for standard Illumina forked adaptors, which have a single 3 thymidine overhang. After adaptor ligation, the BrdU-substituted DNA strands are nicked by photolytic cleavage. This is accomplished by treating samples with Hoechst 33258, followed by UV radiation, which induces a single-stranded nick at sites of BrdU incorporation. Nicking the BrdU+ strand interferes with subsequent PCR amplification, and as a result, only the BrdU– (i.e., the original DNA template) strand is amplified to produce a directional single-cell library. During PCR amplification, a custom multiplexing PCR primer is added to the reaction, such that each cell receives a unique hexamer barcode. This barcode allows multiple cells to be pooled for gel size selection and sequencing. Following Illumina HTS, the template strands in each library are distinguished by the first sequencing read generated from the A2 Illumina adaptor, and the single-cell barcode is read using a custom sequencing primer generated by the second sequencing read. By demultiplexing and aligning sequencing data to a reference assembly, template strand identity is determined for each chromosome in the cell (Fig. 2.9b).

The key difference between Strand-seq and other single-cell sequencing protocols is that it aims to sequence only one strand of DNA in each cell, by isolating daughter cells after one cell division. While many steps mirror other single-cell sequencing methodologies, the ability to produce directional single-strand libraries is dependent on nicking the BrdU-incorporated strands through incubation with Hoechst and subsequent UV irradiation. In addition to introducing a photolytic cleavage step, the Strand-seq protocol also bypasses any whole-genome amplification in order to preserve the specific labeling of the DNA strands. This means that genomic libraries are generated from only 6–10 pg of input DNA. Because of this, and the multiple enzymatic cleanup steps that are required during library preparation, we typically obtain genomic coverage in the range of a few percentage points (1–5%)—depending on the number of libraries pooled and the depth of sequencing achieved. The Strand-seq technology sacrifices coverage (depth and breadth) for long-range structural information that offers many new applications for genome biology, as discussed below. Finally, because the resulting

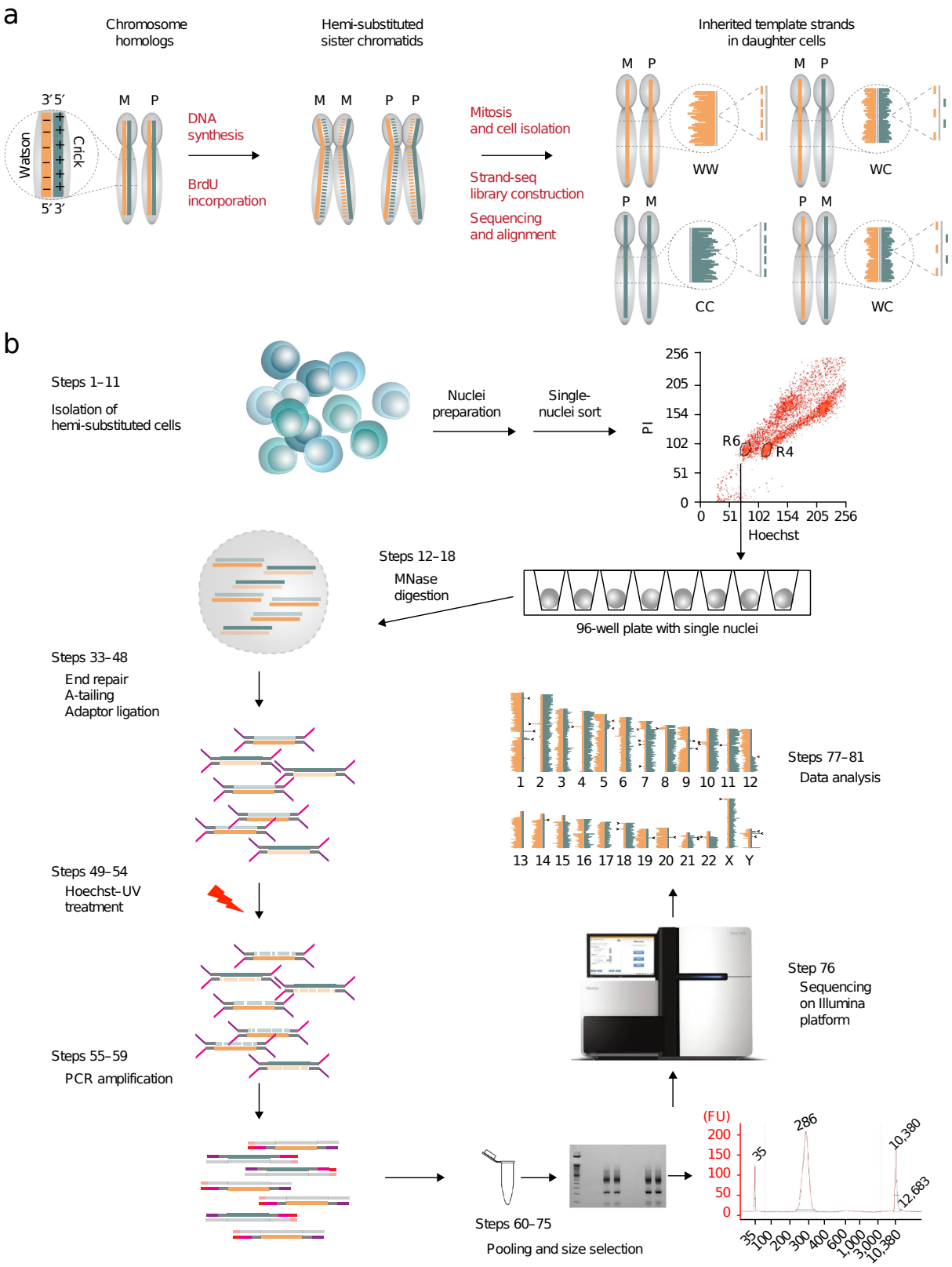


Figure 2.9: Overview of Strand-Seq sequencing protocol.

sequencing data are directional, bioinformatic analysis must consider whether output reads map to the plus or the minus strand of the reference genome, meaning most standard genomic analysis methods cannot be immediately applied to the data.

2.2 Algorithmic Background

Broadly, the algorithms to solve different computational problems are categoried into the following categories based on the type of the solution.

1. Parameterized or FPT algorithms
2. Approximation algorithms
3. Randomized algorithms

2.2.1 Parameterized Algorithms

Imagine that you are an exceptionally tech-savvy security guard of a bar in an undisclosed small town on the west coast of Norway. Every Friday, half of the inhabitants of the town go out, and the bar you work at is well known for its nightly brawls. This of course results in an excessive amount of work for you; having to throw out intoxicated guests is tedious and rather unpleasant labor. Thus you decide to take preemptive measures. As the town is small, you know everyone in it, and you also know who will be likely to fight with whom if they are admitted to the bar. So you wish to plan ahead, and only admit people if they will not be fighting with anyone else at the bar. At the same time, the management wants to maximize profit and is not too happy if you on any given night reject more than k people at the door. Thus, you are left with the following optimization problem. You have a list of all of the n people who will come to the bar, and for each pair of people a prediction of whether or not they will fight if they both are admitted. You need to figure out whether it is possible to admit everyone except for at most k troublemakers, such that no fight breaks out among the admitted guests. Let us call this problem the Bar Fight Prevention problem. Figure 2.10 shows an instance of the problem and a solution for k = 3. One can easily check that this instance has no solution with k = 2.

Efficient algorithms for BAR FIGHT PREVENTION Unfortunately, the bar fight prevention is a classic NP-Complete problem (the read might have heard it under the name VERTEX COVER), and so the best way to solve the problem is by trying all possibilties, right? If the problem instance is small, for example, $n = 100$ people, then the number of possibilties are 2^{1000} in a brute-force manner. Unfortunately, this program would not finish before the guests arrive. There is a good news that the number k of guests that should be rejected is not that large, $k \leq 10$. In this case, the total number of possibilties are $\binom{1000}{10}$. This is better compared to the earlier approach, but this infeasible to do it even on supercomputers.

Let's try different way, how about we identify some peaceful souls to accept, and some troublemakers we need to refuse at the door for sure. Another cases could be identifying those persons who do not conflict. The other case could be identifying a person who fight with atleast $k + 1$ other guests. If you identify such a person, for example, person D in this example, we wanna strick him out, therefore, reducing the number k of people you can reject by one.

Proceeding based on above observation, if there is no such person left, then we know that each guest will fight at most k other guests. Thus, rejecting any single guest will resolve at most k potential conflicts. Also, if there are more than k^2 potential conflict, then there is no way to ensure a peaceful night at the bar by rejecting only k guests at the door. So the guests belong to one of the two categories, one those participating in atleast one and other is at most k potential conflicts, therefore there are at most k^2 potential conflicts, there are at most $2k^2$ guests whose fate is not yet decided. Now, if we try all the possibilties of $\binom{2k^2}{k}$ also takes quite long time even on supercomputers.

After all these cases, it turns out that a simple observation yields a feasible algorithm for BAR FIGHT PREVENTION. The crucial point 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 participants. The way we can solve this problem is as follows. Let's say there is atleast one unresolved conflict between person X and Y. Trying moving X to the reject list and run the algorithm recursively by rejecting at most $k - 1$ guests. If this succeeds, we are done. In case it fails, then move X back to the undecided list and move Y to the reject list and run

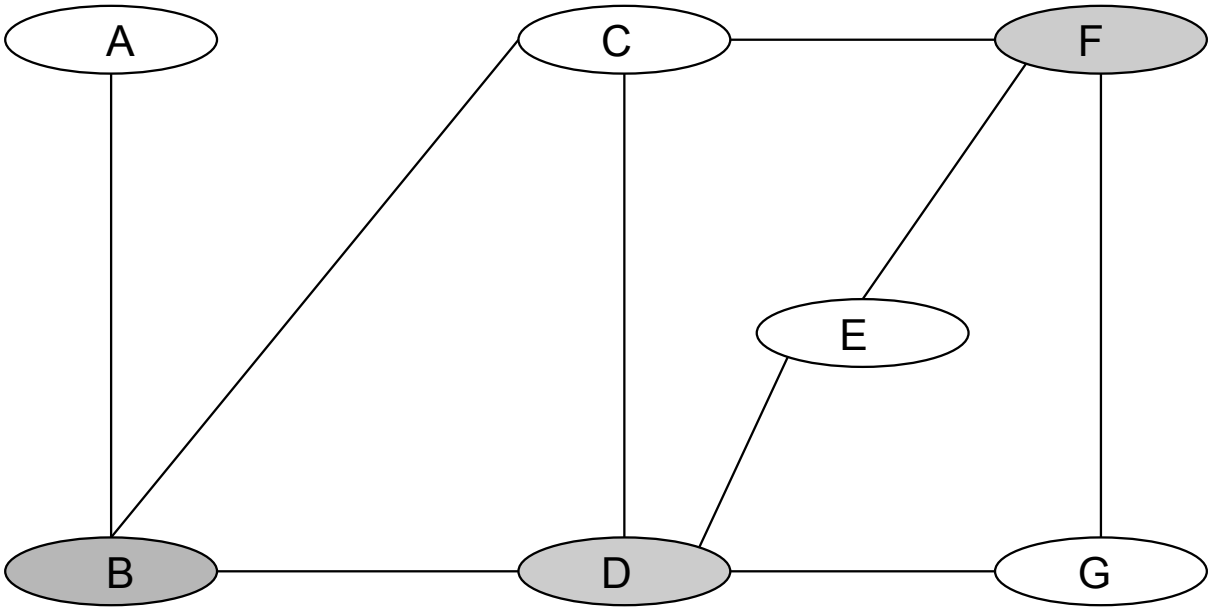


Figure 2.10: An instance of the Bar Fight Prevention problem with a solution for k = 3. An edge between two guests means that they will fight if both are admitted

the algorithm recursively to check whether the remaining conflicts can be resolved by rejecting at most $k-1$ additional guests. If this recursion fails, then we can not solve this instance by rejecting at most k guests.

Time complexity. There are a total of 2^k recursion calls and 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 algorithm is even feasible on a laptop.

The above algorithm runs in time $O(2^k \cdot k \cdot n)$, while the brute-force takes $O(n^k)$ time. There is quite some drastic difference between the running times of both the algorithms.

In $O(2^k \cdot k \cdot n)$— time algorithm, the combinatorial explosion is restricted to the parameter k, the running time is exponential in k, but linear in n. Our goal is to find algorithms of this form.

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 algorithms* or 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 characterics of the input instance.

2.2.1.1 The art of parameterization

For most of the biological problems, we model them as combinatorial optimization problems. We frame an algorithm to solve them using relevant parameters as explained in previous section. For some examples, it is easy to find a parameters from the problem input instance, while it is very difficult for others and require some important insights. For example, consider the variant of BAR FIGHT PREVENTION problem where we want to reject at most k guests such that the number of conflicts (as we believe that the bouncers at the bar can handle l conflicts, but not more). Then we can parameterize either by k or by l. We may even parameterize by both: then 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. In this way, the theory of parameterization and FPT algorithms can be extended to considering a set of parameters at the same time.

Definition 2.2 (Parameterized algorithm (more than one parameter)). Formally, however, one can express parameterization by k and l simply 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.

We can now formulate the extended BAR FIGHT PREVENTION in terms of parameters k and l as follows. These parameters are explicity given in the input, defining properties of the solution we are looking for.

We can have more variants of BAR FIGHT PREVENTION problem, where we need to reject at most k guests such that, say, the numbers of conflicts decreases by p, or such that each accepted guest has conflicts with at most d other accepted guests, or such that the average number of conflicts per guest is at most a. Then the parameters p, d, a are again explicity given in the input, telling us what kind of solution we need to find.

For the string or sequence problems related to genomics, one can parameterize by the maximum read-length, by the maximum coverage, by the size of alphabet, by the number of alleles in a variant.

Parameterized complexity allows us to study how different parameters influence the complexity of the problem. A successful parameterization of a problem needs to satisfy two properties: First, there should be specific reason for the choice of parameters for different applications. Second, we need efficient algorithms where the combinatorial explosion is restricted to the parameter(s), that is, we want the problem to be in FPT with this parameterization.

In conclusion, for the same problem, there can be multiple choices of parameters. Selecting the right parameter(s) for a particular problem is an art.

We now introduce the formal foundation of parameterized complexity.

Definition 2.3. A parameterized problem is a language $L \in \sum * \times N$, where $\sum$ is a fixed, finite alphabet. For an instance $(x, k) \in \sum * \times N$, k is called the parameter.

We define the size of an instance (x, k) of a parameterized problem as $|x| + k$.

Definition 2.4. A parameterized problem $L \subset \sum * \times N$ is called *fixed-parameter tractable* (FPT) if there exists an algorithm $\mathcal{A}$ (called a fixed-parameter algorithm), a computable function $f : N \to N$ and a constant c such that, given $(x, k) \in \sum * \times N$, the algorithm $\mathcal{A}$ correctly decides whether $(x, k) \in L$ in time bounded by $f(x).|(x, k)|^c$. The complexity class containing all fixed-parameter 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 running time has to be of the form of $f(k) \cdot n^c$, one can:

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

2.2.2 Randomized Algorithms

We define a randomized algorithm as an algorithm that is allowed access to a source of independent, unbiased, random bits; it is then permitted to use these random bits to influence its computation. It is easy to sample a random element from a set S by choosing $O(log|S|)$ random bits and then using these bits to index an elememnt in the set.

There are two principal advantages to randomized algorithms. The first is performance — for many problems, randomized algorithms run faster than the best known deterministic algorithms. Second, many randomized algorithms are simpler to describe and implement than deterministic algorithms of comparable performance.

We define a very useful tool from probability theory that is used in the analysis of randomized algorithm: *linear of expectation.* For random variable, $X_1, X_2 \ldots$

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

Let us consider the example of *binary planar partition* of a set of n disjoint line segments in the plane. A binary planar partition consists of a binary tree together with some additional information. Every internal node of a tree has two children. Associated with each node v of a tree in a region $r(v)$ of the plane. Associated with each internal node v of a tree is a line $l(v)$ that intersects $r(v)$. The region corresponding to the root is the entire plane. The region $r(v)$ is partitioned by $l(v)$ into two regions $r_1(v)$ and $r_2(v)$, which are the regions associated with the two children of v. Thus, any region r of the partition is bounded by the partition lines on the path from the root to the node corresponding to r in the tree.

Given a set $S = \{s_1, s_2, \ldots, s_n\}$ of non-intersecting line segments in the plane, we wish to find a binary planar partition such that every region in the partition contains at most one line segment. Notice that the definition allows us to divide the input line segment s_i into several segments $s_{i1}, s_{i2}, \ldots$, each of which lies in a different region. The example shows such a partition of a set into three line segments in Figure 2.11.

For a line segment s, let $l(s)$ denote the line obtained by extending s on both sides to infinity. For the set $S = \{s_1, s_2, \ldots, s_n\}$ of line segments, a simple and natural class of partition is the set of autopartitions, which are formed by only using lines from the set $\{l(s_1), l(s_2), \ldots, l(s_n)\}$ in constructing the partition.

In the algorithm, the input is the set $S = \{s_1, s_2, \ldots, s_n\}$ of non-intersecting line segments. The output is the binary autopartitions of S. To solve it, we pick a permutation π of $\{1, 2, \ldots, n\}$ uniformly at random from the $n!$ possible permutations. Then we cut it with $l(s_i)$ where i is first in the ordering π such that s_i cuts the region. We repeat this process till a region contains more than one segment.

For the analysis of randomized algorithm, we show that the expected size of autopartitions produced by the algorithm is $O(nlogn)$. We can easily show this using linear of expectation of the intersections.

2.2.3 Approximation Algorithms

Most of the real-world problems can be modelled as optimization problems and unfortunately, most interesting discrete optimization problems are NP-hard. Thus unless P=NP, there are no efficient algorithms to find optimal solutions to such problems. We define an efficient algorithm is the one that is solved in time polynomial in input size. What should we do next? How about we find a near-optimal solution which can be found in polynomial time? Furthermore, we focus on finding polynomial-time algorithms for some special cases of the problem, instead of any instance.

Here, we present the *approximation algorithms* for discrete optimization problems. We try to find a solution that closely approximates the optimal solution in terms of its *value*. We assume that there is some *objective function* mapping each possible solution of an optimization problem to some nonnegative value, and an *optimal solution* to the optimization problem is the one that either minimizes or maximizes the value of this objective function. We define an approximation algorithm as follows.

Definition 2.5. An α-approximation algorithm for an optimization problem is a polynomial-time algorithm that for all instances of the problem produces a solution whose value is within a factor α of the value of an optimal solution.

For an α-approximation algorithm, we will call α the *performance guarantee* of the algorithm.

There are several advantages of devising approximation algorithm as follows.

1. An approximation algorithm provides a way to find the near-optimal solutions when the optimal solution is not required and NP hard to find.
2. It give us an idea about how to devise an heuristic that will perform well in practice for the actual problem.
3. It provides a mathematically rigorous basis on which to study heuristics.
4. The field of approximation algorithms gives us a means of distinguising between various optimization problems in terms of how well they can be approximated.

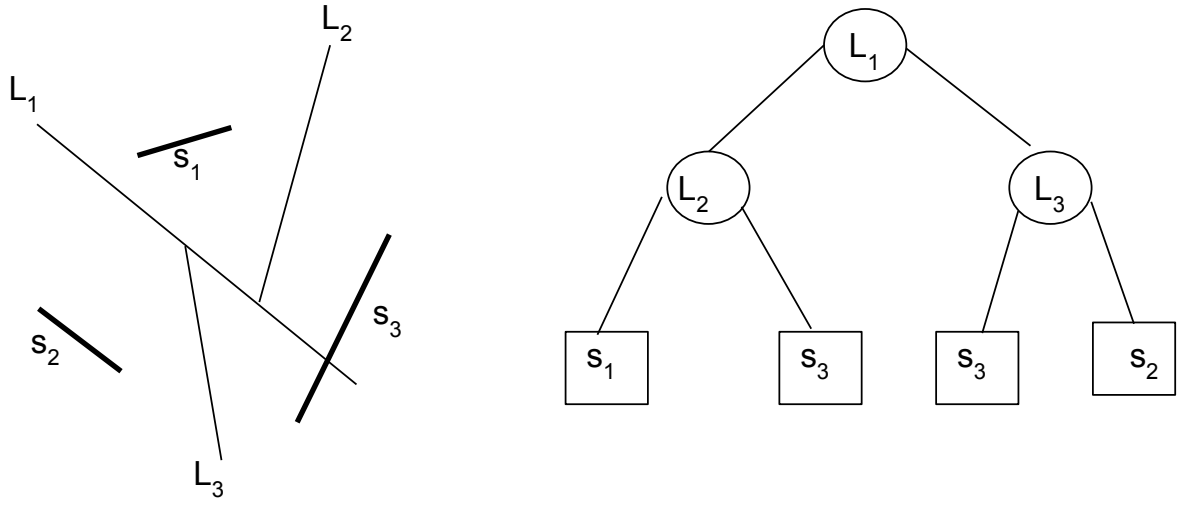


Figure 2.11: An example of a binary planar partition for a set of segments (dark lines). Each leaf is labeled by the line segment it contains. The labels r(v) are omitted for clarity.

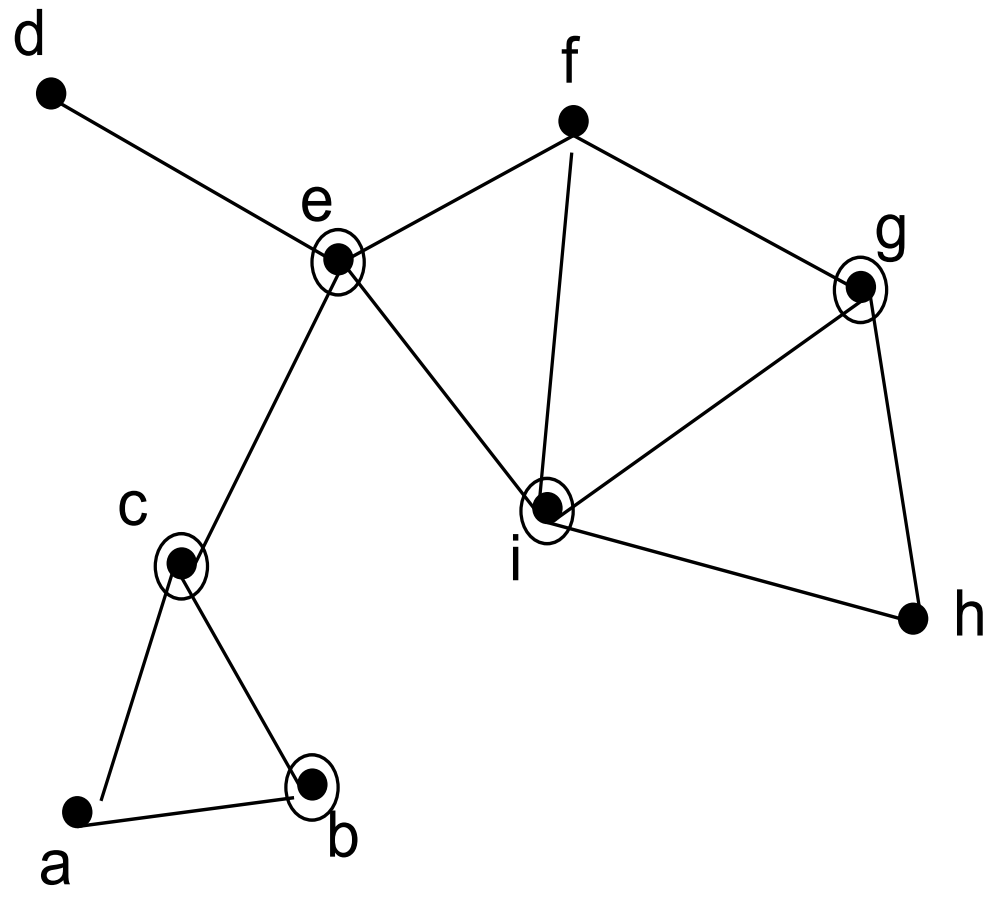


Figure 2.12: An instance of Vertex Cover problem. An optimal vertex cover is b, c, e, i, g.

The next question is about the quality of approximation algorithms, which means does the problems of our interest have *polynomial-time approximation schemes.*

Definition 2.6. A polynomial-time approximation schemes (PTAS) is a family of algorithm $\{\mathcal{A}_\varepsilon\}$, where there is an algorithm for each $\varepsilon > 0$, such that A_ε is a $(1+\varepsilon)$-approximation algorithm (for minimization

problems) or a $(1 - \varepsilon)$-approximation algorithms (for minimization problems).

Let us consider an example of vertex cover in Figure 2.12 to explain approximation algorithm. Given a $G = (V, E)$, find a minimum subset $C \subset V$, such that C *covers* all edges in E, i.e., every edge $\in E$ is incident to at least one vertex in C.

We consider the following steps to solve this problem. Let us consider set C as empty. We pick any edge $\{u, v\} \in E$ and add it to the set C. Afterwards we delete all the edges incident to either u or v. We repeat this process till E is not empty.

The above algorithm is a 2-approximation for vertex cover problem.

Chapter **3**

Parameterized algorithms for individual genomes

We introduced parameterized algorithms in chapter**[TODO: chapter 1]** with a real-world toy example, which provides a good basis for haplotype phasing problem. In this chapter, we provide a integrative framework based on parameterized algorithm for solving the phasing problem for individual genomes. We apply parameterized algorithm on the combination of global, but sparse haplotypes obtained from strand-specific single cell sequencing (Strand-seq) with dense, yet local, haplotype information available through long-read or linked-read sequencing. This results in dense and accurate chromosome-length haplotypes at reasonable costs. We provide comprehensive guidance on the required sequencing depths and reliably assign more than 95% of alleles (NA12878) to their parental haplotypes using as few as 10 Strand-seq libraries in combination with 10-fold coverage PacBio data or, alternatively, 10X Genomics linked-read sequencing data.

3.1 The power of combining different sequencing technologies for phasing

Constructing genome-wide chromosome-length haplotypes is important for different biological applications**[TODO: citation]**. Currently, methods used to chart the unique variation of individual human genomes rely largely on 2nd and 3rd generation DNA sequencing and can include specialized experimental protocols**[TODO: citation]**. Sequencing technologies sample the human genome in the form of relatively short molecules (reads) and every read that spans at least two heterozygous variants can essentially be considered as a 'mini haplotype' that can be assembled into longer haplotype segments by partially overlapping reads spanning the same variable locus**[TODO: citations]**. To this end, haplotype-informative reads need to be partitioned into two disjoint sets that represent the two haplotypes. This process, however, is complicated by errors in sequencing as well as genotyping. For these reasons assembling haplotypes directly from sequencing data is computationally challenging, and the resulting optimization problems are provenly hard**[TODO: citation]**. Notwithstanding, a number of computational approaches for read-based phasing have recently been developed**[TODO: citation]** and, particularly, progress on fixed-parameter tractable (FPT) algorithms has enabled solving read-based phasing in practice**[TODO: citation]**, for instance through the implementations available in the software package WhatsHap**[TODO: citation]**. **[TODO: related work for MEC is missing.]** However, all approaches to reconstruct haplotypes from sequencing reads, be it 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**[TODO: citation]** and Oxford NanoPore MinION**[TODO: citation]**), or linked read data (such as those provided by 10X Genomics**[TODO: citation]**) 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**[TODO: citation]**. As an alternative to whole chromosome separation, chromatin capture (Hi-C) methods**[TODO: citation]** 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**[TODO: citation]**. Recently, Hi-C data have been used in combination with other sequencing methods for long-range phasing**[TODO: citation]**. 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 crosslinks between non-homologues chromosomes**[TODO: citation]**. 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.

Here, we introduce a strategy 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)**[TODO: citation]**. 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**[TODO: citation]**. 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 designed a novel unified statistical 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**[TODO: citation]**. 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.

[TODO: different examples of the genomics regions to show the importance of combining technologies.]

3.2 Integrative phasing framework

We present the integrative phasing method that is a framework to solve phasing jointly using different sequencing datasets presented in Section**[TODO: citation]**. To solve it, we generalize the input instances given in **[TODO: Definiton?]** to jointly include the read alignment or initial haplotypes from different technologies. For example, the haplotypes are generated using strand-specific cells or 10x Genomics, are added to the matrix $\mathcal{F}$. Furthermore, the read-alignments over the variants using different technologies such as PacBio or Illumina are then additionally incorporated in $\mathcal{F}$. The goal is to partition the rows in $\mathcal{F}$ into two non-conflicting sets. The input matrix and the bipartition of the rows is illustrated in Fig. 3.2.

Mathematically, aligned reads from Illumina or PacBio (or pre-phased 10X Genomics haplotype segments) and sparse Strand-seq haplotypes are jointly represented in the form of a fragment matrix, where each row represent either one reads (in case of Illumina and PacBio), one pre-phased haplotype segment (in case of 10X Genomics) or one sparse global haplotype (in case of StrandSeq data) and columns represent the variant sites (Fig. 3.2). 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). 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**[TODO: citation]**.

The definitions **[TODO: conflict-free, bipartition]** are analogous.

The goal is the bipartition of the rows and generate two haplotypes for diploid genomes.

[TODO: explain MEC matrix and its goal using example above.]

[TODO: 1. present MEC matrix example for different technologies like pacbio, illumina, 10xG haps, SS haps. 2. Now state the common observation from all the examples. 3. Then provide a intuition to solve it. 4. Finally provide the DP algorithm like in book DPChange of chapter 6.]

3.2.1 StrandPhaseR pipeline

To build whole genome haplotypes from Strand-seq data we developed a new sorting-based pipeline, called StrandPhaseR. StrandPhaseR implements an improved phasing algorithm based on a binary sorting strategy of two parallel matrices, storing haplotype information obtained from single cell Strand-seq libraries. Haplotype informative WC regions were localized in every Strand-seq library by counting the number of Crick (forward, '+') and Watson (reverse, '-') reads in equally sized regions (default 1 Mb). We used Fisher's exact test to calculate the probability that a region contained approximately equal numbers of Crick and Watson reads and agreed with the expected 50:50 ratio of a WC region**[TODO: citation]**. Alleles at variable positions (supplied as set of SNVs obtained from Illumina platinum haplotypes) were identified separately for W and C reads in every informative region to generate low density single cell haplotypes that are then sorted by the phasing algorithm. The partial single cell haplotypes are used to fill two matrices, where rows represent cells and columns represent covered variable positions (SNVs) in any given cell (**[TODO: Supplementary [TODO: fig]S1]**). Initially, one matrix stores all variable positions found within the Watson templates, and a second matrix stores all variable positions found within the Crick templates. Cells in the matrices are sorted in decreasing order based on the number of covered variants (i.e. depth of coverage). Initially, a score of each column is calculated as the sum of all covered variants minus the most abundant variant. This represents the level of disagreement across all cells for the given SNV in the column. The sum of scores for each column represents the overall score of the matrix, and a lower matrix score represents a higher level of concordance across all SNV positions. Once the score of both matrices is determined, all SNVs in the first row (i.e. those belonging to the first cell) are swapped between the two matrices. In essence, this exchanges the Watson and Crick template strands of the cell within the matrix, to test whether there is a higher level of agreement across the phased SNVs found for all the cells. To determine this, the matrix scores are recalculated and if the scores are lower than the previous scores the change is kept, otherwise the change is reversed. The algorithm continues with the second row. Again, the covered variants of the second cell are swapped between matrices, the matrix score are recalculated and the decision to preserve or reverse the change is made. This is repeated through all rows (cells) of the matrix, sorting the single cell haplotypes within both matrices to reduce the number of conflicting alleles within each column. We repeated sorting process twice, after which we did not observe any further changes. The resulting haplotypes are reported as the consensus allele found across all the cells for each column of the matrices. Ideally, there is only one allele present for every variable site in each matrix, however sporadic sequencing errors or cell-specific artefacts can introduce discrepancies. Lastly, any missing alleles at heterozygous sites are rescued by searching within the 'uninformative' reads (i.e. those from WW and CC regions) present in Strand-seq libraries and filled in.

3.3 Algorithm

Solving MEC and wMEC. WhatsHap (Patterson et al., 2015a) is a dynamic programming (DP) algorithm to optimally solve the wMEC problem. 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 even long sequencing

technologies. 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 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 3.1 (Bipartition extension). For a given set A and a subset $A' \subset A$, a bipartition $B = (P, Q)$ of A is said to *extend* 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.

[TODO: add examples to explain algorithm.]

3.4 Quality metrics of assembled haplotypes

To assess the quality of assembled haplotypes in this study, 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'. 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 that make part of such a 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 on a 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 (**[TODO: Supplementary [TODO: fig]S3aA]**).

Largest haplotype segment: In this study we are interested in haplotypes that span the whole length of all chromosomes. To measure the completeness of phasing, we report the fraction of heterozygous variants that are part of the largest haplotype segment.

Largest haplotype segment Hamming rate: To assess whether haplotypes are correct also over long genomic distances, we only consider the largest haplotype segment and compute the Hamming distance between true and predicted haplotypes (**[TODO: Supplementary [TODO: fig]S3bB]**), 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, only one switch error (e.g. in the middle of a chromosome) can result into a very high 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.

3.5 RESULTS

3.5.1 Publicly available datasets used in this study

[TODO: add this paragraph probably in the text above.] Illumina reads**[TODO: citation]** were obtained from 1000 Genome Project Consortium ([1]). PacBio reads**[TODO: citation]** were downloaded btained from Genome in a Bottle Consortium (GIAB)([2]). 10X Genomics haplotypes: pPre-assembled

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

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

10X Genomics haplotypes (produced on the Chromium plat form 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 ([3]) and filtered for heterozygous and PASS filter SNVs. Strand-seq libraries**[TODO: citation]**: For this study have been downloaded from the European Nucleotide Archive ([4]), accession number: PRJEB14185. Alternatively, aligned Strand-seq data, used in this study, can be obtained at Zenodo site ([5]). Reference haplotypes**[TODO: citation]**: 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) ([6]).

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**[TODO: add data as footnote]**. 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 (**[TODO: Supplementary Table S1]**). In addition, we evaluated the performance of 10X Genomics, an emergent linked-read technology. Since none of these technologies alone provides chromosome-length haplotype information, we additionally incorporated single cell Strand-seq data**[TODO: citation]**, which has the capacity to scaffold haplotype information obtained from other data types (Fig. 3.1(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 (see **[TODO: Data Availability statementccessMethods and Supplementary Table S2]**). 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 (see **[TODO: Data AccessMethods]**). NA12878 'reference haplotypes' were completed by genetic haplotyping using highly accurate genotypes from seventeen individuals of a three-generation pedigree**[TODO: citation]**, 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 (**[TODO: Supplementary Note 1]**). 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).

3.5.2 Downsampling of sequencing datadatasets used in this studyDownsampling of Strand-seq libraries and read data (PacBio or Illumina)

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 (see Data Access) datasets consisting of: 134 single cell Strand-seq libraries**[TODO: citation]**, 39.6x coverage long-read PacBio data**[TODO: citation]**, and 49.6x coverage short-read Illumina data**[TODO: citation]**. 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).

[3] https://support.10Xgenomics.com/genome-exome/datasets/NA12878_WGS_210
[4] http://www.ebi.ac.uk/ena
[5] doi:10.5281/zenodo.830278
[6] http://www.illumina.com/platinumgenomes/

3.6 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 (see Methods). The main advantage of this algorithm is that it solves the Minimum Error Correction (MEC) problem optimally with a runtime 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 10X LongRanger pipeline. To phase multiple Strand-seq libraries we have developed a new phasing algorithm, implemented in the R package StrandPhaseR (**[TODO: see Methods, and Supplementary [TODO: fig]S1]**). In comparison to our previously published phasing algorithm**[TODO: citation]**, the current algorithm is provided as an easy to use R package and implements more robust heuristic approach to solve MEC problem in Strand-seq data. 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 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 to phase 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 (Fig. 3.1(b)). Note that for 10X Genomics data, we used the variant set discovered, genotyped, and phased by 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 (**[TODO: Supplementary [TODO: fig]S2]**). 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 (Fig. 3.1(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 (Fig. 3.1(c)). 10X Genomics produced even longer haplotype segments than both Illumina and PacBio data (Fig. 3.1(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 (Fig. 3.1(c)). That is, the linked reads from 10X Genomics were not able to connect distant neighboring heterozygous sites, for instance at centromeres, genome assembly gaps or regions of low heterozygosity (Fig. 3.1(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 (Fig. 3.1(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 (Fig. 3.1(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.

3.6.1 Integrative global phasing strategy

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 (**[TODO: fig]**3aA). 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 (**[TODO: fig]**3aA, i). 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 (**[TODO: fig]**3aA i, black arrowhead). Similar results were seen when we combined Strand-seq with the 10X Genomics haplotypes (**[TODO: fig]**3aA, ii). In all cases, integration of Strand-seq phasing drastically improved the contiguity of the haplotype spanning Chromosome 1 (**[TODO: fig]**3bB). When combining Illumina data with 40 Strand-seq libraries >65% of the reference variants could be phased accurately (**[TODO: fig]**3bB i, 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 (**[TODO: fig]**3bB ii, 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 (**[TODO: fig]**3bB iii, 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 (**[TODO: fig]**3bB iv). 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) (**[TODO: fig]**3cC). 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 (**[TODO: fig]**3cC, i). We further observed slightly elevated switch error rates at lower PacBio depths, which plateaued at 10-fold coverage (**[TODO: fig]**3Cc, black arrowhead). This is likely caused by allele uncertainty resulting from error-prone PacBio reads, especially at lower sequencing depths (**[TODO: fig]**3C, i). The lowest switch error rate (< 0.2%) was achieved by the combination of Strand-seq with 10X Genomics data (**[TODO: fig]**3cC, ii, 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 (see Methods, Supplementary **[TODO: fig]**S4aA). 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 (see Methods, Supplementary **[TODO: fig]**S4bB). Illumina reads are highly accurate and therefore we observed lower impact of sequencing depth on the global accuracy of the largest phased haplotypes (**[TODO: fig]**3cC, Hamming error rateiii). 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 (**[TODO: fig]**3cC, iii black arrowheadgray rectangle), 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 (**[TODO: fig]**3cC, Hamming error rateiv).

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 (**[TODO: fig]**4aA). 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 (**[TODO: fig]**4aA, i). 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 (**[TODO: fig]**4aA, i,ii). 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 (see Data Availability to access phased SNVs for NA12878 using integrative phasing approach presented in this study.).

3.7 DISCUSSION

Strand-seq has been successfully prepared from a wide range of cell types taken from various organisms9,34,37 and is currently being adopted by an increasing number of researchers. The integrative phasing strategy we introduce here 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 (**[TODO: fig]**4b).

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

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

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 paper, 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, which might not be available in all settings, including clinical contexts. 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 information40 (Supplementary Note 3). Strand-seq relies on BrdU incorporation during DNA replication and its use is therefore restricted to dividing cells. To provide long-range phase information for single samples in situations where growing cells are not available, Hi-C can constitute an alternative solution able to yield chromosome-spanning haplotypes32. However, the required coverage, and hence sequencing cost, is considerably higher for Hi-C than for Strand-seq. While the 134 Strand-seq libraries we used here reach a cumulative sequencing coverage of around 5x, markedly higher coverages are needed for Hi-C32.

Our results demonstrate that dense and accurate chromosome-length haplotypes can be generated

at manageable costs. This development brings haplotype-level analyses closer to a routine practice, which can be key for understanding disease phenotypes. We emphasize that the strategy we present here works for single individuals without relying on other family members or statistical inference from haplotype reference panels. In contrast to such population-based phasing approaches, the method we advocate here allows insights into rare and de novo variants and long-range epistatic effects.

Our future efforts will focus on de novo assembly of haplotype resolved-genomes without the alignment to a reference genome. This will provide us with true diploid representations of individual genomes, which will have profound implications to study variability of personal genomes in health and disease.

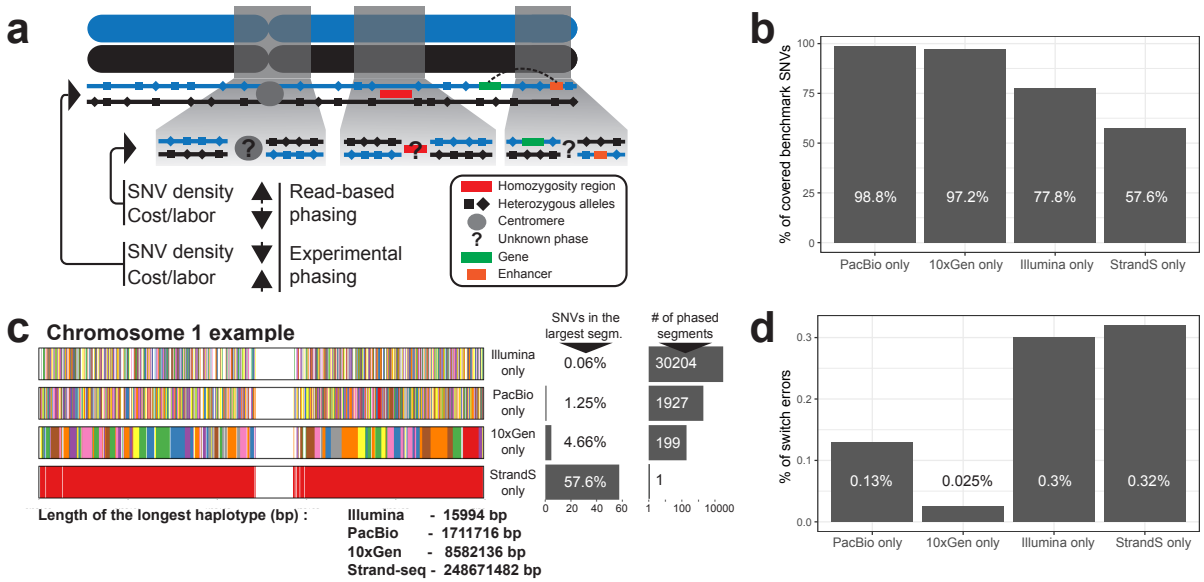


Figure 3.1: Figure 1: Phasing efficacy of read-based and experimental phasing approaches using Chromosome 1 as an (example Chromosome 1). aA) Two homologous chromosomes are shown (blue and black). Experimental phasing approaches like Strand-seq can connect heterozygous alleles along whole chromosomes, 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. bB) Barplot showing the percentage of phased variants, for each sequencing technology, from the total number of reference SNVs (Illumina platinum haplotypes). cC) 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. dD) Accuracy of each independent phasing approach measured as percentage short switch errors in comparison to benchmark haplotypes.

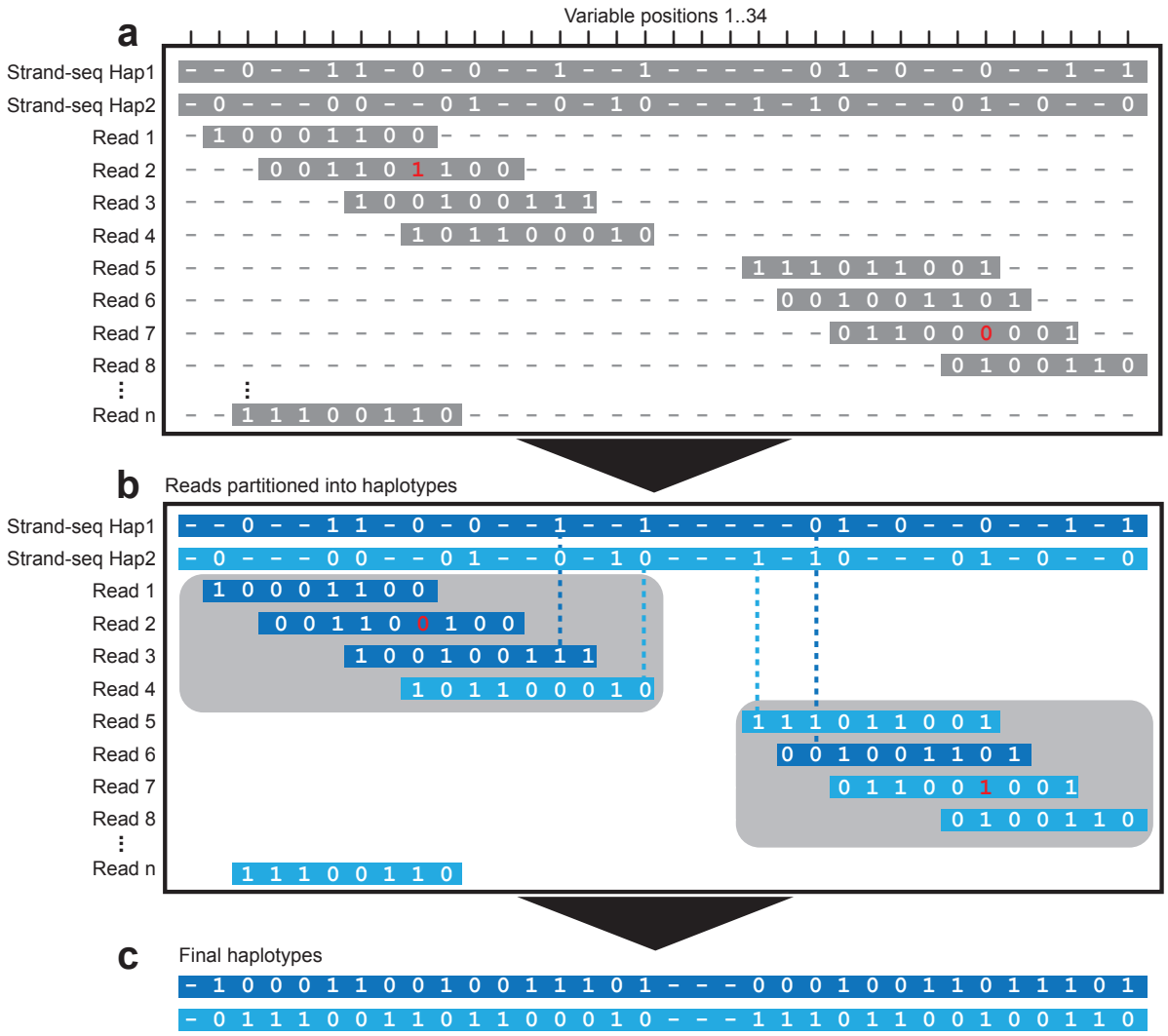


Figure 3.2: Figure 2: Integration of global and local haplotypes by the WhatsHap algorithm. An example solution of the weighted minimal error correction problem (wMEC) using WhatsHap algorithm is shown. For simplicity base qualities used as weights are omitted from the picture (for details on wMEC see Patterson et al. 2015). a(i) 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(ii) Reads are then partitioned into two haplotype groups (Haplotpye 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(iii) Final haplotypes are exported for both groups of optimally partitioned reads. eles. In contrast, read-based phasing can deliver high-density haplotypes, but only short haplotype segments are assembled with an unknown phase between them. bB) Barplot showing the percentage of phased variants, for each sequencing technology, from the total number of reference SNVs (Illumina platinum haplotypes). cC) 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. dD) Accuracy of each independent phasing approach measured as percentage short switch errors in comparison to benchmark haplotypes.

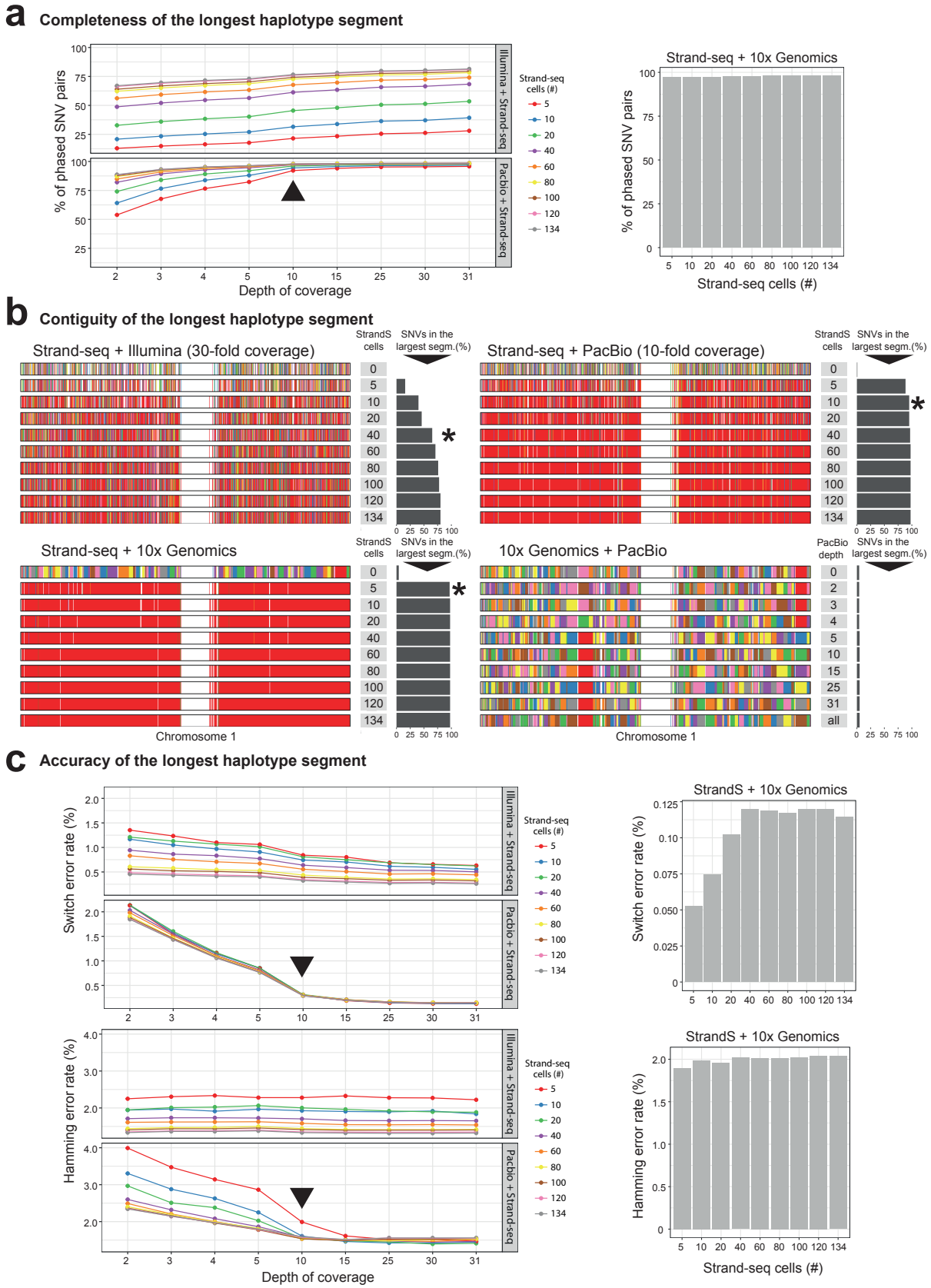


Figure 3.3: Figure 3: Various combinations of Strand-seq and read-based phasing (Illumina, PacBio, 10Xusing Genomics) - example Chromosome 1 as an example. Plots show haplotype quality measures for various combinations of Strand-seq cells (5, 10, 20, 40, 60, 80, 100, 120, 134) with selected coverage depths of Illumina or PacBio sequencing data (2, 3, 4, 5, 10, 15, 25, 30, >30-fold), or in combination with 10X Genomics haplotypes. aA) 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. bB) 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

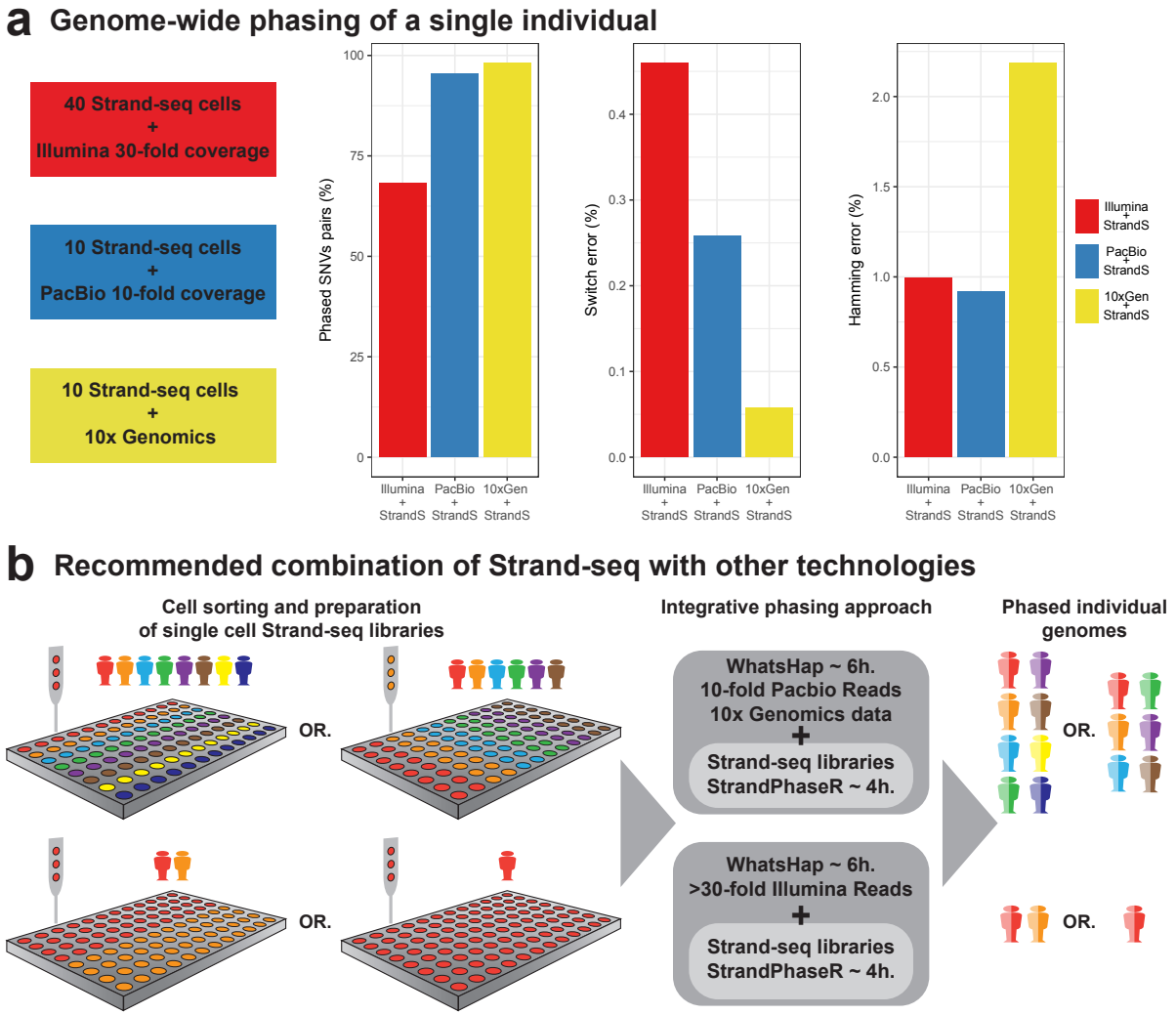


Figure 3.4: Figure 4: Recommended setting to phase certain amount of individuals. aA) Genome-wide phasing of NA12878 using combination of 40 Strand-seq libraries with 30x short Illumina reads, 10 Strand-seq libraries with 10-fold long PacBio reads, or 10 Strand-seq libraries with 10X Genomics data. (i) The percentage of phased SNV pairs in the largest haplotype segment for each combination (ii)The cumulative accuracy the genome-wide haplotype was calculated by summing the switch error rates and (iii) the Hamming error rates found for each autosomes.Plots show quality measures such as percentage of phased SNV pairs, switch error rate and Hamming error rate for phased autosomal chromosomes. bB) A diagram providing the recommendations for the required number of Strand-seq libraries to be combined with recommended minimum of 10-fold PacBio and 30x Illumina coverage in order to reach global and accurate haplotypes for a depicted number of individual diploid genomes.

Chapter **4**

Approximation algorithm for phasing

We consider the problem Minimum Error Correction (MEC). 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.

Gapless-MEC is a relevant problem in computational biology, and it is closely related to segmentation problems that were introduced by [Kleinberg–Papadimitriou–Raghavan STOC'98] in the context of data mining.

Without restrictions, it is known to be UG-hard 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. 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).

4.1 Introduction.

The minimum error correction problem (MEC) is a segmentation problem where we have to partition a set of length m strings into two classes. A MEC instance is given by a set of n strings over $\{0, 1, -\}$ of length m, where the symbol "$-$" is a wildcard symbol. The strings are represented by an $n \times m$ matrix M, where the ith string determines the ith row $M_{i,*}$ of M.

The distance dist of two symbols a, a' from $\{0, 1, -\}$ is

$$\text{dist}(a, a') := \begin{cases} 1: & a = 0, a' = 1 \text{ or } a = 1, a' = 0 \\ 0: & \text{otherwise.} \end{cases}$$

For two strings s, s' from $\{0, 1, -\}^m$ where s_j, s'_j denotes the j-th symbol of the respective string, $\text{dist}(s, s') := \sum_{j=1}^{m} \text{dist}(s_j, s'_j)$. A feasible solution to the MEC is a pair of two strings σ, σ' from $\{0, 1\}^m$. The optimization goal is to find a feasible solution (σ, σ') that minimizes

$$\text{cost}(\sigma, \sigma') := \sum_{i=1}^{n} \min\{\text{dist}(M_{i,*}, \sigma), \text{dist}(M_{i,*}, \sigma')\}.$$

A MEC instance is called *gapless* if in each of the n rows of M, all entries from $\{0, 1\}$ are consecutive. (As regular expression, a valid row is a word of length m from the language $-^*\{0, 1\}^*-^*$). The MEC problem restricted to gapless instances is Gapless-MEC.

Our motivation to study Gapless-MEC stems from its applications in computational biology. Humans are diploid, and hence there exist two versions of each chromosome. Determining the DNA sequences of these two chromosomal copies – called haplotypes – is important for many applications ranging from population history to clinical questions Snyder et al. (2015a); Tewhey et al. (2011a). Many important biological phenomena such as compound heterozygosity, allele-specific events like DNA methylation or gene expression can only be studied when haplotype-resolved genomes are available Leung et al. (2015).

Existing sequencing technologies cannot read a chromosome from start to end, but instead deliver small pieces of the sequences (called reads). Like in a jigsaw puzzle, the underlying genome sequences are reconstructed from the reads by finding the overlaps between them.

The upcoming next-generation sequencing technologies (e.g., Pacific Biosciences) have made the production of relatively long continuous sequences with sequencing errors feasible, where the sequences come from both copies of chromosome. These sequences are aligned to a reference genome or to a structure called contig. We can formulate the result of this process as a Gapless-MEC instance.

We can therefore formulate haplotyping as the problem of reconstructing the genomes from sequences represented in the form of Gapless-MEC instances by correcting the sequencing errors. The use of sequencing datasets for complete haplotype phasing is quickly becoming reality Chin et al. (2016); Chaisson et al. (2015).

Gapless-MEC is a generalization of a problem called Binary-MEC, the version of MEC with only instances M where all entries of M are in $\{0, 1\}$. Finding an optimal solution to Binary-MEC is equivalent to solving the hypercube 2-segmentation problem (H2S) which was introduced by Kleinberg, Papadimitriou, and Raghavan Kleinberg et al. (1998, 2004) and which is known to be NP-hard Feige (2014); Kleinberg et al. (2004). The optimization version of Binary-MEC differs from H2S in that we minimize the number of mismatches instead of maximizing the number of matches. Binary-MEC allows for good approximations. Ostravsky and Rabiny Ostrovsky and Rabani (2002) obtained a PTAS for Binary-MEC based on random embeddings. Building on the work of Li et al. Li et al. (2002), Jiao et al. Jiao et al. (2004) presented a deterministic PTAS for Binary-MEC.

Gapless-MEC was shown to be NP-hard by Cilibrasi et al. Cilibrasi et al. (2007b).[1] Additionally, they showed that allowing a single gap in each strings renders the problem APX-hard. More recently,

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

Bonizzoni et al. 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. To our knowledge, previous to our result their logarithmic factor approximation was also the best known approximation algorithm for Gapless-MEC.

4.1.1 Our results.

We provide a quasi-polynomial time approximation scheme (QPTAS) for Gapless-MEC and thus partially settle the approximability for this problem: Gapless-MEC is not APX-hard unless NP $\subseteq$ QP (cf. Remy and Steger (2009)). 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.

Furthermore, 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. We provide a PTAS for subinterval-free instances.

4.1.2 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 continuous binary parts of the rows. For each length class we consider a well-selected set of columns such that each row crosses at least one the columns and at most two. (We say that a row crosses a column if 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. 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. We can arrange the rows and columns of these instances in such a way that all rows cross the first column. We order the rows from top to bottom by increasing length.

The first level of our DP solves these highly structured instances. The basic idea that we want to apply is that we want to select a constant number of rows from the instance that represents the solution. 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 idea above works.

This insight motivates to separate the instance from left to right into sub-problems 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 its solution 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 build up the DP by considering the smallest units and assambling them to bigger units, i.e., we proceed in reverse order of the separation into sub-instances described above. In order to have a

clear presentation of the result, we first show, in Section 4.2, that it is possible to compute a $(1 + O(\varepsilon))$-approximate Binary-MEC solution without looking at an ε fraction of rows. We then generalize these results in order to be able to handle the sub-instances that we will solve repeatedly in our DP, in Section 4.2.1.

In order to combine the sub-instances, we face technical complications due to having distinct sub-problems for those rows assigned to σ and those rows assigned to σ'. In order to separate these complications from the actual combination of sub-problems, we first consider instances for which we only want to have one solution string, in Section 4.2.2. We note that these instance would be easily solvable by simply selecting the majority value in each column, but our aim is something different. As needed for the actual instances, in the DP we restrict our view and only considering constantly many rows at a time. Afterwards, in Section 4.2.3, we show how to combine our insights with taking care of both σ and σ'.

Before we consider rooted instances, we first develop our techniques by considering subinterval-free instances which are easier to handle (Section 4.3). 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.

Until this point, all of our algorithms are running in polynomial time. When considering arbitrary rooted instances, in Section 4.4, the left hand side of the root may have a completely different structure than the right hand side of the root. 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 sub-problems. Considering all of the sub-problems 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.

We next consider Gapless-MEC instances where all strings are in the same length class, i.e., the longest binary part is at most twice as long as the shortest binary part of a row (Section 4.4.1). As described earlier, we consider two sets of rows such that for each of these sets there is a sequence of columns such that each row crosses exactly one of the columns. 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 4.4.2, we are able to solve all length classes simultaneously by noting that the total number of different length classes is logarithmic in the input. 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 use $\log(n)$-vectors of quadruples. This adds another power of $\log(n)$ to the running time, which is still quasi-polynomial.

4.1.3 Further related work.

Binary-MEC is a variant of the Hamming k-Median Clustering Problem when $k = 2$ and there are PTASs known Jiao et al. (2004); Ostrovsky and Rabani (2002). Li, Ma, and Wang 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.

Alon Alon and Sudakov (1999) provided a PTAS for H2S, the maximization version of Binary-MEC and Wulff, Urner and Ben-David Wulff et al. (2013) showed that there is also a PTAS for the maximization version of MEC. For Gapless-MEC, He et al. He et al. (2010b) 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. (2015b); Garg et al. (2016) (and some additional parameters for pedigrees).

Bonizzoni et al. Bonizzoni et al. (2016) provided FPT algorithms parameterized by the fragment length and the total number of corrections for Gapless-MEC. There are some tools which can be used in practice to solve Gapless-MEC instances Pirola et al. (2015b); Patterson et al. (2015b).

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

Lancia et al. Lancia et al. (2001a) 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.

4.1.4 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.

4.2 All rows cross column one.

Suppose we are given an instance of Gapless-MEC where all entries of column one in M are zero or one. Formally, we consider Gapless-MEC with the restriction that in M, $M_{i,1} \in \{0,1\}$ for each index i. We order the rows 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.

We first build the tools that we need for the simplified setting of Binary-MEC instances. We sometimes consider the majority string of a matrix. The function Majority computes a string of length m by setting the jth entry to the most frequent entry of $M_{*,j}$, for each column j.

In order to compute Binary-MEC solutions, we consider the algorithm Binary$_\delta$ which is based on the PTAS for Binary-MEC obtained by Jiao et al. Jiao et al. (2004). We state Binary$_\delta$ as a randomized algorithm in order to reveal the core ideas in a simple form. Due to the manner in which we want to use the algorithm later on, we introduce parameters R, r, R', r'. R and R' are sets of rows from which the algorithm is allowed to sample. The other two parameters r and r' are integers. We ensure in the algorithm that the computed solution has $|\sigma(M)| = r$ and $|\sigma'(M)| = r'$. The intuition is that we guess $|\tau(M)|$ and $|\tau'(M)|$ and set r, r' to these values. We note that the randomized algorithm Binary$_\delta$ requires the knowledge of an optimal solution (τ,τ').

Input : Two sets of selected rows R, R' from a Binary-MEC instance M, two numbers r, r'.
Output: A pair of strings (σ,σ'), an assignment of the rows to these strings.

1 Select uniformly at random (with repetition) a multi-set $\tilde{R}$ of $1/\delta$ strings from $R \cap \tau(M)$
2 and a multi-set $\tilde{R}'$ of $1/\delta$ strings from $R' \cap \tau'(M)$;
3 Set σ = Majority$(\tilde{R})$ and σ' = Majority$(\tilde{R}')$;
4 For each row i, determine the value $d_i := \mathrm{dist}(\sigma, M_{i,*}) - \mathrm{dist}(\sigma', M_{i,*})$;
5 Assign the r rows with maximal values d_i to σ and the remaining r' rows to σ'.

Algorithm 1: Binary$_\delta$

A simple derandomization helps us to find a solution without knowledge of (τ,τ'): We replace the random selection of $1/\delta$ rows by trying all of the $O(n^{2/\delta})$ possible selections of two row sets $\tilde{R}, \tilde{R}'$ of size

$1/\delta$ each. For each selection, we compute $\sigma = \textsc{Majority}(\tilde{R})$ and $\sigma' = \textsc{Majority}(\tilde{R}')$. We keep the solution (σ, σ') with minimal value $\text{cost}(\sigma, \sigma')$. Since there are only polynomial many possibilities, the algorithm runs in polynomial time. We observe that $\textsc{Binary}_\delta$ with input matrix M and $R = R' = M$, $r = |\tau(M)|$, $r' = |\tau'(M)|$ computes a Binary-MEC solution to M; here we can guess the values $|\tau(M)|, |\tau'(M)|$ by trying all possibilities.

Lemma 4.1. For $r = |\tau(M)|$ and $r' = |\tau'(M)|$ let R, R' be sets of rows such that $|\tau(R)| \geq (1-\varepsilon)r$ and $|\tau'(R')| \geq (1-\varepsilon)r'$. Then the derandomized version of $\textsc{Binary}_\varepsilon$ is a $(1 + O(\varepsilon))$-approximation algorithm for Binary-MEC.

Lemma 4.1 is a special case of Lemma 4.3. We therefore postpone the proof.

We are now ready to continue with a first (simplified) version of a technical key lemma which we will refer to as "partial input lemma." Intuitively, the lemma states that we can find a $(1 + O(\varepsilon))$ approximation for Binary-MEC, even if a fraction of rows (depending on ε) is hidden.

Lemma 4.2. For a Binary-MEC instance M and sufficiently small $\varepsilon > 0$, let R be an arbitrary set of rows from M with at least $(1-\varepsilon)|\tau(M)|$ rows from $\tau(M)$ and R' an arbitrary set of rows from M with at least $(1-\varepsilon)|\tau'(M)|$ rows from $\tau'(M)$. Let $r = |\tau(R)|$ and $r' = |\tau(R')|$. Let (σ, σ') be the solution computed by $\textsc{Binary}_\varepsilon$ for the matrix $P = R \cup R'$. Then (σ, σ') is a $(1 + O(\varepsilon))$-approximation for the instance M.

Let $Q := M \setminus P$ where we identify M with its set of rows. The claim still holds if within Q, we assign $\tau(Q)$ to σ and $\tau'(Q)$ to σ'.

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. The matrix M has n rows and m columns. Let η be the total number of errors of (τ, τ') within M and let η' be the total number of errors of (σ, σ') within P. Clearly $\eta' \leq (1 + O(\varepsilon))\eta$. 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 $|\sigma(P) \cap \tau(P)| \geq n/6$.

Claim 4.1. There is a set I of $m - 25\eta/n$ indices i such that $\sigma_i = \tau_i$ for all $i \in I$.

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 $12(1 + O(\varepsilon))\eta'/n < 13\eta/n$ 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 $i \in I$ with $\tau_i \neq \sigma_i$. Then, since $|\tau(P) \cap \sigma(P)| \geq n/6$, either σ_i or τ_i 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 decisions.

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

Claim 4.2. For each column j of I, either $\sigma'_j = \tau'_j$ or $\eta''_j \geq (1-\varepsilon)|\tau'(M)|/2$.

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

Since by our assumption $|\tau'(Q)| < \varepsilon|\tau'(M)|$, Claim 4.2 implies that within I we have a $(1 + O(\varepsilon))$-approximation.

To finish the proof, we argue that η is large enough to pay for all errors in Q outside of I. Due to our discussion regarding σ' in I, it is save to set $\sigma(Q) := \tau(Q)$ and $\sigma'(Q) := \tau'(Q)$. Then (σ, σ') has at most $O(\varepsilon)\eta$ more errors than (τ, τ') within Q for the columns of I. Let this number of errors be η_I.

Then, using the size of I stated in Claim 4.1, the total number of errors of (σ, σ') in M is at most $(1 + O(\varepsilon))\eta + \eta_I + \varepsilon n \cdot 25\eta/n$, i.e., the errors of $\textsc{Binary}_\varepsilon$ within P, the errors within Q in the columns of I, and all other entries of Q. The obtained approximation ratio is

$$\frac{(1 + O(\varepsilon))\eta + \eta_I + \varepsilon n \cdot 25\eta/n}{\eta} \leq \frac{\eta + O(\varepsilon)\eta + 25\varepsilon\eta}{\eta} = 1 + O(\varepsilon)\,.$$

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

4.2.1 Simple instances with wildcards.

Next we show that we can extend the PTAS for Binary-MEC to a specific class of Gapless-MEC instances $\mathcal{G}$. Recall that we consider Gapless-MEC instances M where the first column of M has only binary entries and the rows are ordered in increasing length of the binary part. An instance is in $\mathcal{G}$ if it satisfies these conditions and additionally, 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 $\{0, 1\}$. For an instance $G \in \mathcal{G}$, we consider several sets of consecutive rows. In the following, values r and r' have the same meaning as in $\textsc{Binary}_\delta$.

Let R_A be the set of rows such that (i) all rows with wildcards are contained in R_A, (ii) there are exactly[2] $(1 - \varepsilon)r$ rows from $\tau(G)$ in R_A, (iii) R_A contains the first row, and (iv) R_A is the minimal set with these properties. We added point (iv) merely to avoid ambiguity. The set R_B contains the rows after R_A such that (v) there are $(\varepsilon - \varepsilon^2)r$ rows from $\tau(G)$ in R_B and (vi) R_B is the minimal set with this property. Then the set R_C contains all remaining rows of G. The row sets R_A, R_B, R_C for one solution string are shown in Figure 2. We define R'_A, R'_B, and R'_C analogously, but replace r and τ by r' and τ'. We assume that all considered terms are integers.

We introduce a new algorithm SWC_δ (Simple Wildcards) for our setting. The algorithm is similar to $\textsc{Binary}_\delta$: we consider two row sets R and R' and sample rows from these sets. But additionally, we partition them into $2/\varepsilon^2$ disjoint subsets. Let $\mathcal{R} := \{R_1, R_2, \ldots, R_{2/\varepsilon^2}\}$ and $\mathcal{R}' := \{R'_1, R'_2, \ldots, R'_{2/\varepsilon^2}\}$. We want that the first $1/\varepsilon^2$ sets of $\mathcal{R}$ and $\mathcal{R}'$ are partitions of R_A and R'_A, and the second $1/\varepsilon^2$ sets are partitions of R_B and R'_B. For a multiset of rows S let $\textsc{Majority}_j(S)$ be the *weighted* majority restricted to column j defined as follows. We only consider sets of rows R_i such that in column j, all entries of R_i are in $\{0, 1\}$; otherwise we remove all copies of rows in $R_i \cap S$ from S. For $\ell := 1/\varepsilon^2$ let $U_j := S \cap \bigcup_{i \leq \ell} R_i$ and $L_j = S \cap \bigcup_{i > \ell} R_i$. In U_j and L_j, we replace all zeros by -1 and compute the number $\nu := \sum_{i \in U_j}(1 - \varepsilon)i/(\varepsilon - \varepsilon^2) + \sum_{i \in L_j} i$. Then $\textsc{Majority}_j(S) = 0$ if $\nu < 0$ and $\textsc{Majority}_j(S) = 1$ if $\nu \geq 0$. The intuitive meaning is that we sample 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 sampled with the same marginal probability. Let (τ, τ') be an optimal solution.

We derandomize SWC_δ analogous to $\textsc{Binary}_\delta$. Again, the knowledge of (τ, τ') is required in the randomized algorithm but not in the derandomized one.

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 $\tau(G)$ if and only if we know $\tau'(G)$, and we are able to guess constantly many rows.

Lemma 4.3. For sufficiently large $r = |\tau(G)|$ and $r' = |\tau'(G)|$, let $R = R_A \cup R_B$ and $R' = R'_A \cup R'_B$ be sets of rows as described above. For $\ell = \varepsilon^{-2}$, let $\mathcal{R} = \{R_1, R_2, \ldots, R_{2\ell}\}$ and $\mathcal{R}' = \{R'_1, R'_2, \ldots, R'_{2\ell}\}$ be partitions of R, R' such that (i) for each set R_i with $i \leq \ell$, $|\tau(R_i)| = \varepsilon^2 \cdot (1 - \varepsilon)r$; (ii) for each set R_i with $i > \ell$, $|\tau(R_i)| = \varepsilon^2 \cdot (\varepsilon - \varepsilon^2)r$; (iii) for each set R'_i with $i \leq \ell$, $|\tau'(R'_i)| = \varepsilon^2 \cdot (1 - \varepsilon)r'$; and (iv) for each set R'_i with $i > \ell$, $|\tau'(R'_i)| = \varepsilon^2 \cdot (\varepsilon - \varepsilon^2)r'$.

[2] To be precise, this assumption introduces a small imprecision which needs careful but straightforward handling.

Input : An instance $G \in \mathcal{G}$, partitions $\mathcal{R}$ and $\mathcal{R}'$ of row sets R and R' into $2/\varepsilon^2$ classes, numbers r, r'.
Output: A pair of strings (σ, σ').

1 For each i, we consider $R_i \in \mathcal{R}, R'_i \in \mathcal{R}'$. Select uniformly at random (with repetition) a multi-set $\tilde{R}_i$ of $1/\delta$ strings from each $R_i \cap \tau(G)$ and a multi-set $\tilde{R}'_i$ of $1/\delta$ strings from $R'_i \cap \tau'(G)$;
2 Set $\tilde{R} := \bigcup_i \tilde{R}_i$ and $\tilde{R}' := \bigcup_i \tilde{R}'_i$; `// Note that` $\tilde{R} \cap \tilde{R}' = \emptyset$.
3 For each column j, set $\sigma_j := \textsc{Majority}_j(\tilde{R})$ and $\sigma'_j := \textsc{Majority}_j(\tilde{R}')$;
4 For each row i, determine the value $d_i := \mathrm{dist}(\sigma, G_{i,*}) - \mathrm{dist}(\sigma', G_{i,*})$;
5 Assign the r rows with maximal values d_i to σ and the remaining r' rows to σ'.

Algorithm 2: SWC_δ

Then the derandomized version of $\mathrm{SWC}_{\varepsilon^3}$ is a $(1 + O(\varepsilon))$-approximation algorithm for instances from $\mathcal{G}$.

The proof is based on using Chernoff bounds and can be found in Appendix .1. With this preparation we are able to generalize Lemma 4.2 to work with SWC_δ instead of $\textsc{Binary}_\delta$. We obtain a generalized version of the partial input lemma.

Lemma 4.4. For a Gapless-MEC instance $M \in \mathcal{G}$ and sufficiently small $\varepsilon > 0$, let R be an arbitrary set of rows from M with at least $(1 - \varepsilon^2)|\tau(M)|$ rows from $\tau(M)$ and R' an arbitrary set of rows from M with at least $(1 - \varepsilon^2)|\tau'(M)|$ rows from $\tau'(M)$. Let $r = |\tau(M)|$, $r' = |\tau(M')|$, and $\mathcal{R}, \mathcal{R}'$ as required in Lemma 4.3. Let (σ, σ') be the solution computed by $\mathrm{SWC}_{\varepsilon^3}$ for the matrix $P = R \cup R'$. Then (σ, σ') is a $(1 + O(\varepsilon))$-approximation for the instance M.

Let $Q := M \setminus P$ where we identify M with its set of rows. The claim still holds if within Q, we assign $\tau(Q)$ to σ and $\tau'(Q)$ to σ'.

The new setup only requires small changes in the proof of Lemma 4.2. Instead of the entire instance M, for the analysis we have to focus on the sub-matrix without $R_A \cap R'_A$. All remaining changes are straightforward.

For the dynamic program, we need that the selection of rows withing R_B provides an interface between two sub-instances. The following lemma follows directly from our previous analyses.

Lemma 4.5. Let us consider a run of $\mathrm{SWC}_{\varepsilon^3}$ with the parameters of Lemma 4.4, but instead of a random sampling in R_B, R'_B, we consider an arbitrary feasible selection of rows $\tilde{R} \cap R_B, \tilde{R}' \cup R_B$. Suppose that with this selection we obtain an α-approximation for P, then we obtain an $\alpha + O(\varepsilon)$-approximation for M.

4.2.2 A simplified DP for a single solution string.

To show how to use Lemma 4.4 in our DP, we first focus on how to determine one of the two solution strings. To this end we assume that we already know $\tau(M)$ and $\tau'(M)$ and that in $\mathrm{SWC}_{\varepsilon^3}$ we have $\sigma' = \tau'$. We restrict our access to these sets, however, to only considering a constant number of rows at a time. Let $r = |\tau(M)|$ and $r' = |\tau(M)|$.

We design a dynamic program with DP cells for each combination of the following parameters. (a) Three consecutive ranges of rows determined by numbers $1 \leq a < b < c < d \leq n$. These numbers determine an upper range R_A from row a to row $b - 1$, a lower range R_B from row b to row $c - 1$ and a bottom range R_C from c to d. (b) A multiset T_U of $1/\varepsilon^5$ rows from R_A. (c) A multiset T_L of $1/\varepsilon^5$ rows from R_B. (d) A partition of T_U into $1/\varepsilon^2$ ranges of rows, determined by numbers $a = a_1 < a_2 < \cdots < a_{1/\varepsilon^2} < b$. (e) A partition of T_L into $1/\varepsilon^2$ ranges of rows, determined by numbers $b = b_1 < b_2 < \cdots < b_{1/\varepsilon^2} < c$. Among these choices, we only consider cells where each range in (d) and (e) contains $1/\varepsilon^3$ rows from T_U resp. T_L.

For each DP cell ζ we store a string σ^ζ from column 1 to the end of the first string of R_B, i.e., row b of M. (Recall that the rows are ordered in increasing length from top to bottom.)

The intuitive meaning of a DP cell is that the first class R_A contains a $1 - \varepsilon$ fraction of rows assigned to τ within the considered range and R_B contains an $\varepsilon - \varepsilon^2$ fraction. The selected rows of T_U and T_L

should be thought of as a random selection of rows from $\tau(M)$ (with repetition) such that in each range from (c) and (d) there are $1/\varepsilon^3$ sampled rows.

A DP cell corresponds to an instance $D_\zeta = R_A, R_B, R_C$ for which we only consider the columns before the end of the bth row.

The algorithm works in two phases. The first phase is an initialization. We assign an initial DP value to *each* DP cell. The value specifies the approximate number of errors if the DP cell is the first one that contributes to the solution (with smallest value a). Afterwards, we will update the initial values in a second phase of the algorithm.

We initialize each cell ζ by applying Lemma 4.4 on D_ζ, but instead of sampling rows in $\mathrm{SWC}_{\varepsilon^3}$, we use T_U and T_L. The selection of rows takes the role of the derandomization step. The partition $\mathcal{R}$ is determined by (d) and (e). Since we only consider one solution string, we do not have to fix r', R' and $\mathcal{R}'$. The result is the string σ^ζ that does not depend on R_C.

Let B be the sub-matrix of M that contains the rows R_B and all columns where the first row of R_B has entries in $\{0, 1\}$. In order to update the DP values and to analyze the quality of the computed solution we count the number of errors within $\tau(M)$ restricted to the columns of B. For each computed string σ^ζ, the value of its DP cell is the number of errors within $\tau(M)$ between rows a and c.

For all DP cells with $a = 1$ in (a), we already have assigned the final value in the initialization step. We now iteratively update the cells for $a \geq 2$. Let ζ be the cell that we want to update and $\tilde{\zeta}$ another cell with the parameters marked by the symbol prime. We say that $\tilde{\zeta}$ is a *predecessor* of ζ if $a = \tilde{b}$; and $b = \tilde{c}$; and $c = \tilde{d}$; and $T_U = \tilde{T}_L$, where the variables marked with tilde are the parameters of $\tilde{\zeta}$.

To compute ζ, we assume that all of its predecessors are updated already. The DP selects the predecessor $\tilde{\zeta}$ with the minimal DP value (i.e., the minimum number of errors within the prefix). To compute σ^ζ, we consider the columns of D_ζ after $\sigma^{\tilde{\zeta}}$, i.e., exactly those columns for which we have not computed a solution yet. We apply Lemma 4.4 on the obtained instance analogous to the initialization phase. By appending the obtained solution after $\sigma^{\tilde{\zeta}}$ we obtain a new candidate string for σ^ζ. We compare the old value of ζ with the number of errors in D_ζ between a and c. If the candidate string incurs fewer errors than σ^ζ, we update ζ: its value is the new number of errors and we store the candidate string. After considering all predecessors of ζ, we have computed its final value. In particular, now we can also use it as a predecessor of other cells. Eventually all possible update operations are done. 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 $c = d = n$. Intuitively, we use these cells when only $1/\varepsilon^3$ rows of $\tau(M)$ are left. For these cells, our computation considers the optimal solution for the suffix of σ.

Lemma 4.6. If we can determine the number of errors within $M(\tau)$ with respect to each prefix of σ, then the algorithm above is a PTAS.

Proof. Since all binary strings are feasible solutions, our algorithm vacuously produces a valid solution.

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 R_i contains the next $(\varepsilon^i - \varepsilon^{(i+1)})r$ rows of $\tau(M)$. To be consistent with properties needed in later proofs, we ensure that the first row of each R_i is contained in $\tau(M)$ and thus we add the rows between R_i and R_{i+1} to R_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 $1/\varepsilon^3$ rows of $\tau(M)$ are located between a and b and R_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.

For each range we obtain blocks corresponding to those of Figure 3. The block A_0 contains the rows of R_0 and the columns one to the end of the first row of $\tau(R_0)$. For each $i > 0$, block A_i contains the rows of R_i and the columns after those of A_{i-1} to the end of the first row of R_i. Then B_i is composed of the columns of A_i and A_{i+1} and of the rows of A_{i+1}. The block C_i has the same columns as B_i but the rows of A_{i+2}.

We consider the DP cells ζ_i for each i with the following parameters. (a) $\hat{a}_i$ is the first row of A_i, $\hat{b}_i$ the first row of B_i, $\hat{c}_i$ the first row of C_i, and $\hat{d}_i$ the last row of C_i. (b),(c) T_U and T_L are sets of $1/\varepsilon^5$ rows from $\tau(A_i)$ and $\tau(B_i)$, respectively, satisfying the requirements of the DP cells. These rows are selected in such a way that applying $\mathrm{SWC}_{\varepsilon^3}$ they are at most the expected solution value obtained by picking uniformly at random with repetition. (d), (e) the parameters $\hat{a}_{i,j}$ and $\hat{b}_{i,j}$ for $j = 1, 2, \ldots, 1/\varepsilon^3$ partition $\tau(A_i)$ and $\tau(B_i)$ into sets of equal sizes.

We obtain sub-instances D_i for each $i \geq 0$ such that D_i contains all rows from $\hat{a}_i$ on and the columns from the end of row $\hat{a}_i$ to the end of row $\hat{b}_i$ The remaining parameters $r_i, R_i, \mathcal{R}_i$ follow directly from (a) to (e). For each D_i there is exactly one DP cell ζ_i.

We inductively show that the expected value of each cell the value of 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 D_0 and the invariant follows directly from Lemma 4.4.

Suppose now that $i \geq 0$ and for all $\tilde{i} < i$ the invariant is true. Then we consider the subinstance D_i.

We apply Lemma 4.4 to compute the string σ^{ζ_i} for D_i. We obtain a $(1 + \varepsilon)$ for the prefix covered by σ^{ζ_i} for the following reason. The part before D_i was fixed, and independent of the rows from $\hat{b}_i$ on already gave a $(1 + \varepsilon)$ approximation. The part of σ^{ζ_i} within D_i gives a $(1 + \varepsilon)$ approximation by the claim of Lemma 4.4.

Observe that these results hold without considering any row from $\hat{c}_i$ on, and by Lemma 4.5 we only have to consider the rows of D_i to ensure this property. We can therefore continue the induction until the entire string σ is determined. □

4.2.3 Adding the second string.

Next we show how to embed a dynamic program for σ' into the dynamic program for σ. To distinguish the two parts, we call one dynamic program the σ-DP and the other one the σ'-DP.

A σ-DP cell ζ and a σ'-DP cell ζ' have the same parameters as our previous DP. (Again, we mark the parameters of ζ' with the symbol prime). The cell ζ' is a *child* of ζ if $b' > b$ and $T_U \cup T_L$ is disjoint to $T'_U \cup T'_L$. The child relationship is shown in Figure 4. The intuition is that the string $\sigma^{\zeta'}$ is shorter than σ^{ζ}. We define a child of ζ' analogously by changing the roles of σ and σ'. Observe that a σ-DP cell cannot be a child of another σ-DP cell and no σ'-DP cell a child of another σ'-DP cell. The child relationship is orthogonal to to the predecessor relationship defined in Section 4.2.2: a cell ζ can be a child of a cell ζ' even though a predecessor of ζ' is the child of ζ. In contrast to the child-relationship, the predecessor relationship always refers to the same type of cell: either both are cells from the σ-DP or both are cells from the σ'-DP.

The new dynamic program has a *DP*-cell for each pair (ζ, ζ') of a σ cell ζ and each child ζ' of ζ and, conversely, each pair (ζ', ζ) of a σ' cell ζ' and each child ζ of ζ'. Additionally, we store two numbers r and r' that are used to guess $|\tau(M)|$ and $|\tau'(M)|$.

By renaming the strings we may assume that the DP starts with a σ-DP. For each σ-DP cell we run an initialization similar to Section 4.2.2. At this point we cannot compute the values of the σ-DP cell since they depend on σ'. Nevertheless, the content of σ^{ζ} in $\mathrm{SWC}_{\varepsilon^3}$ only depends on ζ and not on ζ'. We can therefore assume σ^{ζ} to be known within the initialization phase. For each cell (ζ, ζ'), we store a pair of solution strings $\rho(\zeta, \zeta')$. These two strings have the same length as $(\sigma^{\zeta}, \sigma^{\zeta'})$, but the content is determined by the DP. The value of a cell (ζ, ζ') is the minimum number of errors when considering $\rho(\zeta, \zeta')$ in all rows of M above b'.

We compute the value and pair of strings of the DP cell (ζ, ζ') as follows. Let us first assume that neither ζ nor ζ' has a predecessor. Then the first string of $\rho(\zeta, \zeta')$ is σ^{ζ} and the second string is $\sigma^{\zeta'}$, as we do not rely on previous DP entries at this point. Let j be the index of the last entry of $\sigma^{\zeta'}$. Then the value of (σ, σ') is the minimum number of errors in the submatrix of M formed by the rows above b' when we assign the strings of $\rho(\zeta, \zeta')$ to these rows optimally. The intuition is that these are the only rows for which we have complete information. At the same time, these are the rows that we want to forget about in the subsequent DP.

We inductively assume that all DP cells for predecessors of ζ' have been computed already. There are two cases how the DP can proceed. For the second iteration of the DP, these cases are shown in Figure 5. We continue with the same ζ. The first case is that we consider a σ'-cell ζ'' that is a child of ζ and has predecessors that are also children of ζ. We assume that for all cells including these predecessors we have computed the final DP value. We determine the infix of σ and σ' after the end of the first row of T''_U until the last column of the last row of T''_U as we did before. Then we consider the predecessors $\tilde{\zeta}$ of ζ'' that are children of ζ. We take the $\tilde{\zeta}$ with minimal DP value of $(\zeta, \tilde{\zeta})$. Our new prefix of σ' is composed of the prefix of $(\zeta, \tilde{\zeta})$ and the new infix. The value of (ζ, ζ'') is the total number of errors above b'', the version of b' for ζ''.

The crux of the DP is the "switch" between σ and σ' that happens in the second case, if the subsequent σ'-cell is not a child of σ. Let ζ'' be a σ'-cell that is not a child of σ. Clearly, every meaningful ζ'' here has the property that $T_U \cup T_L$ is disjoint to $T''_U \cup T''_L$. Thus we require that ζ is a child of ζ''. For each predecessor ζ' of ζ'' such that ζ' is a child of ζ, we compute the infix of σ' after the prefix of σ' determined by (ζ, ζ') until the end of the last row of T''_U. The difference, however, is that we do not know all values of σ: the computed infix of σ' reaches beyond the prefix determined for σ. The solution to the new situation is to switch the role of σ and σ': we now compute the cell (ζ'', ζ). The value of this cell is the minimum number of errors above b over all choices of predecessors ζ'.

Now the case where also ζ has a predecessor follows analogous to the case where ζ' has a predecessor by switching the roles of σ and σ'.

As in the previous section, we consider DP cells with $c = d = n$ in order to terminate the computation correctly.

Theorem 4.1. The combined DP is a PTAS.

Proof. To see that the DP works in polynomial time, we observe that instead of simple DP cells in Lemma 4.6 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 4.6, 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 $(1 - \varepsilon)r$ rows of $\tau(M)$ is below the first row of the last $\varepsilon r'$ rows of $\tau'(M)$. Figuratively this means that in Fig. 4 the bottom of A and the top of B' overlap.

We consider DP cells similar to the proof of Lemma 4.6. Starting from the top-most row of $\tau(M)$, for each $i \geq 0$, the ith range R_i contains the next $(\varepsilon^i - \varepsilon^{i+1})r$ rows of $\tau(M)$. The rows not in $\tau(M)$ can be assigned to the ranges arbitrarily. To make the selection compatible with the guessing of ranges, we ensure that the first row of each R_i is contained in $\tau(M)$. Then we choose R_i such that all rows of M until R_{i+1} are contained in R_i. Again, 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 and b and R_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.

For each range we obtain blocks corresponding to those of Figure 3. The block A_0 contains the rows of R_0 and the columns one to the end of the first row of $\tau(A)$. For each $i > 0$, block A_i contains the rows of R_i and the columns after those of A_{i-1} to the end of the first row of R_i. Then B_i is composed of the columns of A_i and A_{i+1} and of the rows of A_{i+1}. The block C_i has the same columns as B_i but the rows of A_{i+2}.

We consider the DP cells ζ_i for each i with the following parameters. (a) a_i is the first row of A_i, b_i the first row of B_i, c_i the first row of C_i, and d_i the last row of C_i. (b),(c) T_U and T_L are a sets of $1/\varepsilon^5$ rows from $\tau(A_i)$ and $\tau(B_i)$, respectively, chosen in such a way that they match the expected value of a selection chosen uniformly at random with repetition that respects $\mathcal{R}$. (d) the parameters $a_{i,j}$ for $j = 1, 2, \ldots, 1/\varepsilon^3$ partition τi into sets of equal sizes. Analogously we define $a'_i, A'_i, b'_i, B'_i, c'_i, C'_i, d'_i$ for ζ'_i.

We construct a solution SOL and inductively show that the expected 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. 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 ζ'_0 is a child of ζ_0. We apply Lemma 4.4 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 the prefixes is therefore at most a factor $(1 + \varepsilon)$ larger than in (τ, τ').

Next suppose that ζ'_1 is a child of ζ_0. Then, similar to the proof of Lemma 4.6, we apply Lemma 4.4 to obtain $\sigma'^{\zeta'_1}$. Now we argue that the choice of the new DP cell ensures that the new (longer) solution prefixes $\rho(\sigma_0, \sigma'_1)$ again give a $(1+\varepsilon)$-approximation. Since σ^{ζ_0} does not depend on the choices of (ζ_0, ζ'_1), we only have to consider the second string. By Lemma 4.5, the selected rows in B'_1 are sufficient to ensure that we can afford not to look at C'_1. Since the DP cell dictates the choice of rows within A'_1 and B'_1, and σ'_0 is a predecessor of σ'_1 (with the same selection within A'_1), the choice of the predecessor cell can only influence the solution by a factor $(1 + \varepsilon)$. Observe that we consider each column in only a constant number of cells. This ensures that the overall error adds up to at most a factor $(1 + O(\varepsilon))$.

The argument does not depend on σ_0 to be the first cell of the instance and can therefore be iterated. The case with $(\zeta'_i, \zeta_{i'})$ is analogous.

Finally suppose that we have computed the value of $(\zeta_i, \zeta'_{i'})$ and $\zeta'_{i'+1}$ is *not* a child of ζ_i. Observe that now ζ_i is a child of $\zeta'_{i'+1}$. As before, we use Lemma 4.4 to bound the error of $\sigma'^{B'\zeta'_{i'+1}}$. The only difference is that now $\sigma'^{\zeta'_{i'+1}}$ is fixed and we argue about ζ_i as a child. □

4.3 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'. 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.

Lemma 4.7. 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. We order the rows of 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.

Let s and t be the first and the last row of M. The column j determines a block b 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, b has only binary entries.

Observe that 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 4.1. Similarly, 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 4.1 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 4.1 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.

(a) Three consecutive ranges of rows determined by numbers $1 \leq \overleftarrow{d} < \overleftarrow{c} < \overleftarrow{b} < \overrightarrow{b} < \overrightarrow{c} < \overrightarrow{d} \leq n$. These numbers determine an upper range R_A 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 $\overleftarrow{c}+1$ and a left bottom range $\overleftarrow{R}_B$ from $\overleftarrow{c}$ to $\overleftarrow{d}$, as well as a right lower range $\overrightarrow{R}_L$ from row $\overrightarrow{b}$ to row $\overrightarrow{c}-1$ and a right bottom range $\overrightarrow{R}_B$ from $\overrightarrow{c}$ to $\overrightarrow{d}$. (b) A multiset T_U of $1/\varepsilon^5$ rows from R_A. (c) A multiset $\overleftarrow{T}_L$ of $1/\varepsilon^5$ rows from $\overleftarrow{R}_L$. (c') A multiset $\overrightarrow{T}_L$ of $1/\varepsilon^5$ rows from $\overrightarrow{R}_L$. (d) A partition of T_U into $1/\varepsilon^2$ classes, determined by numbers $\overleftarrow{b} = a_1 < a_2 < \cdots < a_{1/\varepsilon^2} < \overrightarrow{b}$. (e) A partition of T_L into $1/\varepsilon^2$ classes, determined by numbers $\overleftarrow{b} = \overleftarrow{b}_1 < \overleftarrow{b}_2 < \cdots < \overleftarrow{b}_{1/\varepsilon^2} < \overleftarrow{c}$. (e') A partition of $\overrightarrow{T}_L$ into $1/\varepsilon^2$ classes, determined by numbers $\overrightarrow{b} = \overrightarrow{b}_1 < \overrightarrow{b}_2 < \cdots < \overrightarrow{b}_{1/\varepsilon^2} < \overrightarrow{c}$.

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.

The cells $(\zeta_W, \zeta'_W) \in \gamma$ takes a special role as common "center" of two separate DPs . Observe that for each feasible entry of (ζ_W, ζ'_W), we can apply Theorem 4.1 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 4.1. The strings only overlap between the columns j_t, j_s where we obtain an instance of Binary-MEC. The two solution strings within this block are determined by the rows from $\overleftarrow{c}$ to $\overrightarrow{c}$ and from $\overleftarrow{c}'$ to $\overrightarrow{c}'$, depending only on (ζ_W, ζ'_W). None of the remaining steps from Section 4.2.3 interfere with each other. We therefore run the following *DP*. We first compute all cells (ζ_W, ζ'_W). For each cell, we store an infix of σ and an infix of σ'. The infix of σ starts at t_j and ends at s_j. The entries of the two strings are those that we obtain from $\mathrm{SWC}_{\varepsilon^3}$. (In this case, it is even sufficient to apply $\textsc{Binary}_{\varepsilon}$.)

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) that simulates a choice of rows that is uniformly at random and ranges satisfying the conditions of Lemma 4.4. By Lemma 4.4, we can ensure 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 4.1. □

We now generalize Lemma 4.7 to 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 column 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 4.7.

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. Consider a block $\overrightarrow{T}$ of M that only contains rows that cross q_i and a block $\overleftarrow{T}$ of M that only contains rows that cross q_{i+1}. We say that $\overrightarrow{T}$ τ-dominates $\overleftarrow{T}$ if for each column c of $\overrightarrow{T}$ with a non-zero number of binary entries of c in $\tau(\overrightarrow{T})$, the number of binary entries in $\tau(\overrightarrow{T})$ is at least $1/\varepsilon$ the number in $\tau(\overleftarrow{T})$. We analogously define the τ-dominance of $\overleftarrow{T}$ and τ'-dominance.

We observe that if $\overrightarrow{T}$ is τ-dominant over $\overleftarrow{T}$ 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 $\tau(\overrightarrow{T})$ increases and the number of binary entries of $\tau(\overleftarrow{T})$ decreases. Analogously, if $\overleftarrow{T}$ is τ-dominant over $\overrightarrow{T}$ 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 $\overrightarrow{T}$ on the left hand side of I are τ-dominant and the columns of $\overleftarrow{T}$ on the right hand side of I are τ-dominant. The different cases for I are shown in Figures 6 and 7.

The *dominance region* of $\overrightarrow{T}$ is the set of columns where $\overrightarrow{T}$ is dominant over $\overleftarrow{T}$, and vice versa. Within

the dominance region, our old DP can simply compute solutions without considering interferences: the dominated block is small enough to be ignored by Lemma 4.4.

Within the interval I, both cells have to "cooperate." Effectively we obtain a Binary-MEC block in the middle with a slope on top and a slope on the bottom. This sub-instance can be solved directly. We handle it similar to the center in Lemma 4.7.

We use DP cells similar to Lemma 4.7, but with the following differences. For some j, let us consider column $q_j \in q$. Then we consider such a collection κ_j of special DP cells $(\zeta_{q_j}, \zeta'_{q_j}) \in \kappa_j$, with the same properties as (ζ_W, ζ'_W). We refer to them as the jth center cells. Additionally, for each center cell we also store the dominance regions on the left and right of q_j (i.e., we guess the boundaries of the left and the right interval I). Formally this means to extend the cells by for numbers that encode two intervals $\overleftarrow{I}$ and $\overrightarrow{I}$. For each of the regions, the cells also store the intervals in the same manner as before: numbers $\overleftarrow{c}_1, \overleftarrow{c}_2, \ldots, \overleftarrow{c}_{1/\varepsilon^2}$ that partition those strings crossing q_j that reach into I but do not necessarily span over I, numbers $\overleftarrow{d}_1, \overleftarrow{d}_2, \ldots, \overleftarrow{d}_{1/\varepsilon^2}$ of the remaining rows that entirely span over I, and the same type of partition $\overrightarrow{c}_1, \overrightarrow{c}_2, \ldots, \overrightarrow{c}_{1/\varepsilon^2}$ and $\overrightarrow{d}_1, \overrightarrow{d}_2, \ldots, \overrightarrow{d}_{1/\varepsilon^2}$ for $\overrightarrow{I}$. For each of these row ranges, the cell contains a selection of $1/\varepsilon^3$ rows. These values all belong to ζ_{q_j} and we add the same type of values for ζ'_{q_j}.

We now determine an infix of σ and σ' at $\overrightarrow{I}$ and $\overleftarrow{I}$ by applying $\mathrm{SWC}_{\varepsilon^3}$, again with the random selection replaced by the DP selection.

We embed our old DP into an outer DP that processes the entries of q from left to right. For each cell $(\zeta_{q_1}, \zeta' q_1)$, we compute the prefixes of σ, σ' until q_1 exactly as in Lemma 4.7. 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 $\overrightarrow{I}$, we use the already fixed choice instead of guessing new rows.

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.

Theorem 4.2. Let M be a Gapless-MEC instance such that no string is the substring of another string. Then the DP above is a PTAS for M.

Proof. Let (τ, τ') be an optimal solution. For each separate $q_j \in q$, we run the same DP as in Lemma 4.7 and thus we obtain a $1+\varepsilon$-approximation. For the intervals $\overleftarrow{I}$ and $\overrightarrow{I}$, there is a choice of parameters that matches the choices analyzed in Lemma 4.7. 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 regions without dominance. Then the number of errors that the computed solution (σ, σ') has within this region is at most the number of errors in the solution of $\mathrm{SWC}_{\varepsilon^3}$, and we apply Lemma 4.4 with an empty ignored region to show that also the in-fixes of (σ, σ') at I give a $(1+\varepsilon)$-approximation, analogous to the centers in Lemma 4.7. □

4.4 A QPTAS for general instances.

In the previous section, we crucially used that there are specific columns with the property that we could identify a sub-problem belonging to the left hand side and distinct sub-problem belonging to the right hand side. The intersection of both sides was taken care of using a single DP cell which was enough to separate both sides. Without the non-inclusion property, we lose this nice structure.

The issue already appears in the special case of *rooted* instances (shown in Figure 8(a)) where there is a single column c (the root) such that each string of the instance crosses c. We can solve rooted Gapless-MEC, however, if we allow quasi-polynomial running time.

We apply two distinct orderings of rows to obtain our sub-problems. We first order the rows with increasing starting positions as in Lemma 4.7. An optimal solution (τ, τ') then determines a sequence of

sub-matrices $\overleftarrow{A}_1, \overleftarrow{A}_2, \ldots, \overleftarrow{A}_k$. Instead of moving from $\overleftarrow{A}_1$ to $\overleftarrow{A}_k$ using a DP, however, we now guess the strings for all k sub-matrices *simultaneously*. Additionally, we guess the matrices $\overleftarrow{A}'_1, \overleftarrow{A}'_2, \ldots, \overleftarrow{A}'_{k'}$ simultaneously. We obtain a combined DP cell $\overleftarrow{\zeta}$ for $k + k'$ sub-matrices. Afterwards we reorder the rows in order to handle the right hand side. More precisely, we order the strings in increasing position of the *last* binary entry. The obtained structure corresponds to the right hand side of the instance handled in Lemma 4.7. Again we form the sub-matrices $\overrightarrow{A}_1, \overrightarrow{A}_2, \ldots, \overrightarrow{A}_{k'}$ analogous to the left hand side and guess the selected strings of all matrices simultaneously, giving a combined DP cell for all matrices on the right hand side.

A DP cell for the left hand side is compatible to a DP cell on the right hand side, if there is no string assigned to σ on the one side and to σ' on the other side. The algorithm keeps the solution with fewest errors over all combinations of a left DP cell with a compatible right DP cell.

Lemma 4.8. The algorithm above is a QPTAS for the rooted case of Gapless-MEC.

Proof. To analyze the running time, we observe that k and k' are at most $\log_{1/\varepsilon}(n)$ 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 $\log_{1/\varepsilon}(n)$ 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[3].

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 4.7, 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 4.7 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 4.7, 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. □

4.4.1 Length classes.

We next show how to use Lemma 4.8 to 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 L_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 4.9. For each $i \geq 0$ there is a set $q_i = \{q_{i,1}, q_{i,2}, \ldots\}$ of columns such that (a) each string in L_i crosses at least one column from q_i and (b) no string from L_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 $k \cdot m/2^{i+1}$. Observe that the distance between two consecutive columns from q_i is $m/2^{i+1}$, which matches the shortest length of strings in L_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 L_i is at most $m/2^i$. Let j be an index. The distances 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 $q_{i,j+2}$ as shown in Figure 8(b).

[3]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.

The last claimed property follows directly from the construction of the sets q_i as shown in Figure 8(b). □

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

We first show that we can handle each class separately.

Lemma 4.10. There is a QPTAS for Gapless-MEC if all strings are in the same class L'_i or L''_i.

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

For each column $q_{i,j}$, we create a set of DP cells that stores information about a center region and about non-domination intervals, exactly as in the proof of Theorem 4.2. We can do this by using different row orderings on the left hand side and on the right hand side of $q_{i,j}$, as we did in the proof of Lemma 4.8. For each $q_{i,j}$, let ξ_j be the set of super-cells from Lemma 4.8 for $q_{i,j}$, but with the additional center and non-domination information. 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 4.2, but we consider the left hand side and right hand side of each cell from ξ_j simultaneously. The analysis is the same as in Lemma 4.8. □

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

Lemma 4.11. There is a QPTAS for Gapless-MEC if all strings are in the same class L_i.

Proof. To combine the PTAS for L'_i and L''_i, we proceed from left to right. For each index j let ξ'_j be the sets of DP cells for L'_i and for the odd indices j let ξ''_j be the set of cells for L''_i. For a column c before $q_{i,1}$, each pair of cells $(C, C') \in \xi'_1 \times \xi''_1$) determines two subproblems (sets of boxes) that are used for computing the two separate solutions. Let $\mathcal{B}(C, C', z)$ be the set of all boxes (sets of rows) for σ considered in the sub-cells of (C, C') at column z and let $\mathcal{B}'(C, C', z)$ be the set of all boxes (sets of rows) for σ' considered in the sub-cells of (C, C') at column z. We can interpret Lemma 4.4 such that there is a (C, C') for which the chosen rows determine the solutions of the subproblems for c entirely.

We now extend the DP as follows. We compose solution from left to right, starting with the prefix of (σ, σ') before $q_{i,1}$ and then step by step filling 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 some 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 L'_i and L''_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 vectors from the proof of Lemma 4.3 that contain those rows from $\tau(M)$ that belong to L'_i resp. L''_i and are contained in sets from $\mathcal{B}(C, C', \ell)$ resp. $\mathcal{B}'(C, C', \ell)$ with only binary entries. Then the entry σ_ℓ is the weighted majority of entries. The weighted majority is defined as in the algorithm $\mathrm{SWC}_{\varepsilon^3}$, but for both $\hat{c}_1$ and $\hat{c}_2$, i.e., we scale the weight of each entry from $\hat{c}_1$ by $|\hat{c}_1|/(|\hat{c}_1| + |\hat{c}_2|)$ and each entry from $\hat{c}_2$ by $|\hat{c}_2|/(|\hat{c}_1| + |\hat{c}_2|)$. Recall that in the proof of Lemma 4.3 we argued that for each single column the expected number of errors is at most a factor $(1 + \varepsilon)$ larger than for (τ, τ'). We can apply exactly the same argument here, with the only difference that we consider four different density classes instead of two. Again, for τ' the arguments are analogous.

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 $(C''_j, C''_{j+2}) \in \xi''_j \times \xi''_{j+2}$ there are two consecutive pairs of cells $(C'_j, C'_{j+1}) \in \xi'_j \times \xi'_{j+1}$ and $(C'_{j+1}, C'_{j+2}) \in \xi'_{j+1} \times \xi'_{j+2}$. For a quadruple of cells $(C'_j, C''_j, C'_{j+1}, C''_{j+2})$ we consider each quadruple on

the left hand side ending with the matching cells C'_j, C''_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 $(C'_{j+1}, C''_j, C'_{j+2}, C''_{j+2})$, we consider all cells $(C'_j, C''_j, C'_{j+1}, C''_{j+2})$, i.e., the cells that have the same $C''_j, C''_{j+1}, C'_{j+1}$ for all choices of C'_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 along the lines of the analysis in Lemma 4.10 and the quasi-polynomial running time from that we only consider constantly many super-cells simultaneously.

□

4.4.2 The general QPTAS.

Finally we combine our insights to an algorithm for general instances shown in Figure 9.

We partition the rows into their at most $\log_2(m)$ length classes L_i. The main idea is that for each column j, we only have to consider those quadruple of super-cells from Lemma 4.11 that cross j.

We therefore consider at most $O(\log(n))$ quadruples of super-cells simultaneously. 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.

For each length class i, a quadruple $q \in Q_{i,j}$ can 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 $(C'_\ell, C''_\ell, C'_{\ell+1}, C''_{\ell+2})$ (see proof of Lemma 4.10). It ends in j if it is formed by $(C'_{\ell-1}, C''_{\ell-2}, C'_\ell, C''_\ell)$. If j lies between $q_{i,\ell}$ and q_i, ℓ, j crosses those quadruples that contain C'_ℓ and $C'_{\ell+1}$. 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. We require that if for some i, the quadruple $q \in Q_{i,j}$ ends at j, then the same also holds for all quadruples of shorter length classes (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. 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 DP for general instances follows the ideas of Lemma 4.11. We move from left to right column by column. 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' match the left hand sides of the quadruples of v (see proof of Lemma 4.11). Then the value of v is the minimum value over all v'.

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 4.11. Observe that in Lemma 4.11 we combined two pairs of super-cells that we can see as two length classes L' and L''. The only difference is that here we combine $\log(n)$ quadruples of super-cells.

Theorem 4.3. The above algorithm is a QPTAS for Gapless-MEC.

Appendix

.1 Proof of Lemma 4.3

Proof. 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 our discussion about derandomization.

We consider the jth column of G. Let $c := \tau(G)_{*,j}$, 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 $\tau(G)$ 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 R and ignore R_C.

There is a further issue regarding $\mathcal{R}$. Let s be the smallest index such that R_s and c intersect, i.e., R_s is the first set with binary entries in column j. Then rows sampled for R_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 R_i for $i > s$.

To summarize, c has at least εr entries and we ignore at most $2\varepsilon^2$ of these due to R_C and R_s. For each $i > s$, we sample $1/\varepsilon^3$ rows from R_i. Let c' be c restricted to $\bigcup_{s<i\leq\ell} R_i$ and let c'' be c restricted to $\bigcup_{i>\ell} R_i$. For each i, let c_i be the fraction of zeros of c in R_i. Let $\hat{c}$ be c without R_s and R_C and let $\bar{c}$ be the part of c in $R_C \cup R_s$. For each i, we define p_i to be the fraction of zeros c_i.

We define a random variable $X_{i,k}$ for each $s < i \leq 2\ell$ and $1 \leq k \leq 1/\varepsilon^3$. For each i, k, we pick an entry from c_i uniformly at random. Then $X_{i,k}$ is the value of the picked entry. For all i, k, $E[X_{i,k}] = 1 - p_i$. Observe that the $X_{i,k}$ are independent Poisson trials. Let $X' := \sum_{s<i\leq\ell, 1\leq k\leq 1/\varepsilon^3} X_{i,k}$ and $X'' := \sum_{i'>\ell, 1\leq k\leq 1/\varepsilon^3} X_{i,k}$. We want to use Chernoff bounds to control the probability to take the wrong decision. It is sufficient to consider X' with $s = \ell - 1$, since in all other cases the probabilities are amplified more. Let $\mu' := E[X']$. 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

$$\Pr(X' \geq 1/(2\varepsilon^3)) < \left(\frac{e^{\delta'}}{(1+\delta')^{(1+\delta')}}\right)^{\mu'} = \left(\frac{1}{1+\delta'}\right)^{\mu'} \left(\frac{e}{1+\delta'}\right)^{\mu'\delta'} \tag{1}$$

$$= (2\mu'\varepsilon^3)^{\mu'} (e \cdot 2\mu'\varepsilon^3)^{(1/(2\varepsilon^3)-\mu')} \tag{2}$$

Note that both terms of (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 X' and obtain $\bar{X}' := \varepsilon^3 X'$, which has values between zero and one. Then

$$\Pr(\bar{X}' - E[\bar{X}'] \geq \varepsilon) \leq e^{-2\varepsilon^2/\varepsilon^3} = e^{-2/\varepsilon}.$$

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 X' and X'', we introduce a bias for X' such that we count rows i for $s < i \leq \ell$ with a factor $(1 - \varepsilon)/(\varepsilon - \varepsilon^2)$. Then

$$\bar{X} := \frac{\bar{X}' \cdot (\ell - s)(1 - \varepsilon)/(\varepsilon - \varepsilon^2) + \bar{X}'' \cdot \ell}{(\ell - s)(1 - \varepsilon)/(\varepsilon - \varepsilon^2) + \ell}.$$

Then, using the union bound, setting $\sigma_j = 0$ for $\bar{X} < 1/2$ 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 $X_{i,k}$ to be zero or one. Thus the random process used by the algorithm can only have a lower variance.

This finishes our analysis for $\tau(G)_{*,j}$. For $\tau'(G)_{*,j}$, the proof is analogous.

To finish the proof we still have to argue that we may assign the r rows with maximal values d_i to σ and the remaining r' rows to σ'. The reason is that at all times we compare against the assignment of (τ, τ'), which is a stronger result then the mere approximation ratio. Since $|\tau(G)| = r$ and the choice of d_i gives the assignment with fewest errors, the claim follows. □

Again, 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.

.2 Figures

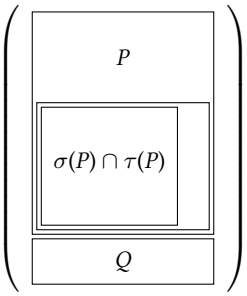


Figure 1: Separation of M into blocks P and Q, where the rows of M are reordered for ease of presentation. The Block on the left hand side shows the rows $\sigma(P) \cap \tau(P)$ within columns I (moved to the left).

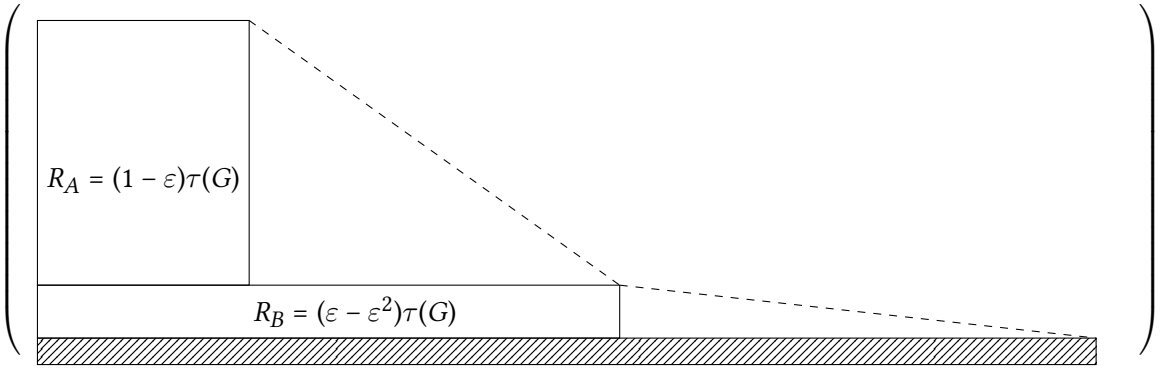


Figure 2: Row sets on an instance G . The bottom stripe depicts a block R_C of G. All the blocks have binary values without wildcards.

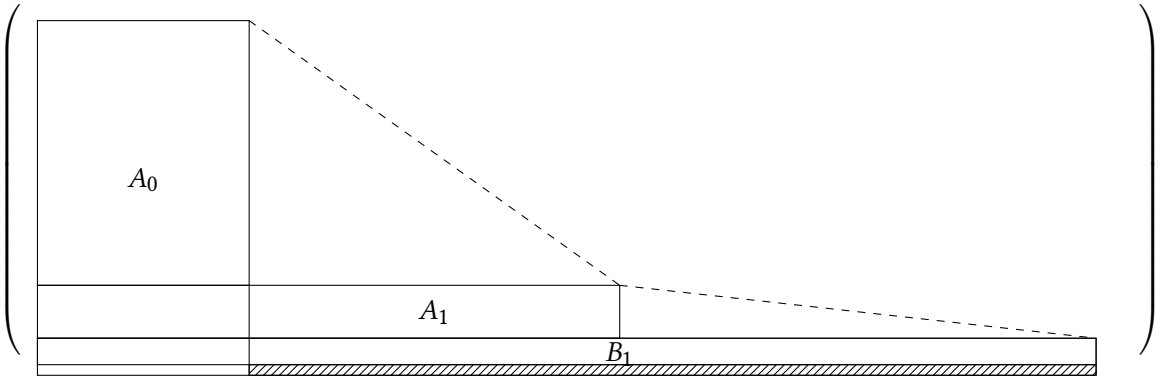


Figure 3: Blocks A_i, B_i on an instance G for the i^{th} iteration of DP in Lemma 4.6. The bottom stripe depicts a block C_1 of G.

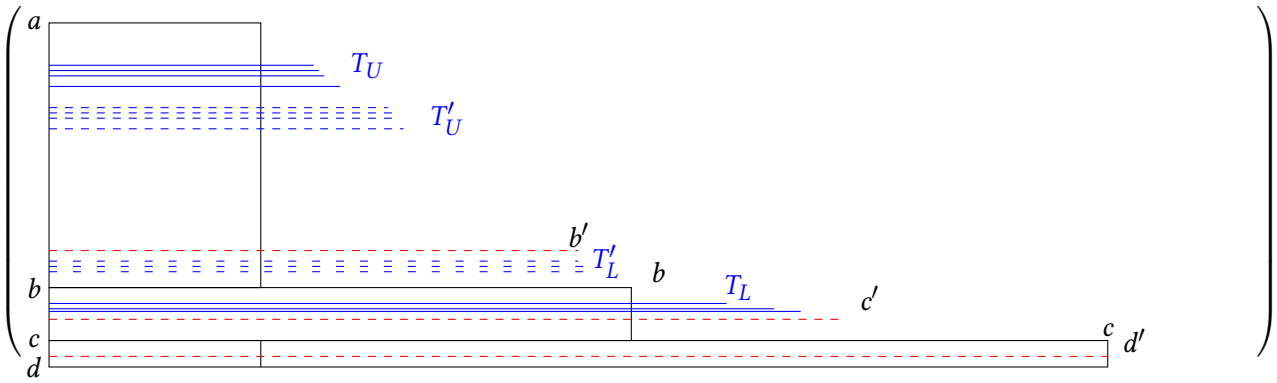


Figure 4: Example of a child cell in the DP for the second string. The cell starting at b' (drawn in red) is a child of the cell staring at a (drawn in blue), because $b' > b$ and $T_U \cup T_L$ is disjoint from $T'_U \cup T'_L$.

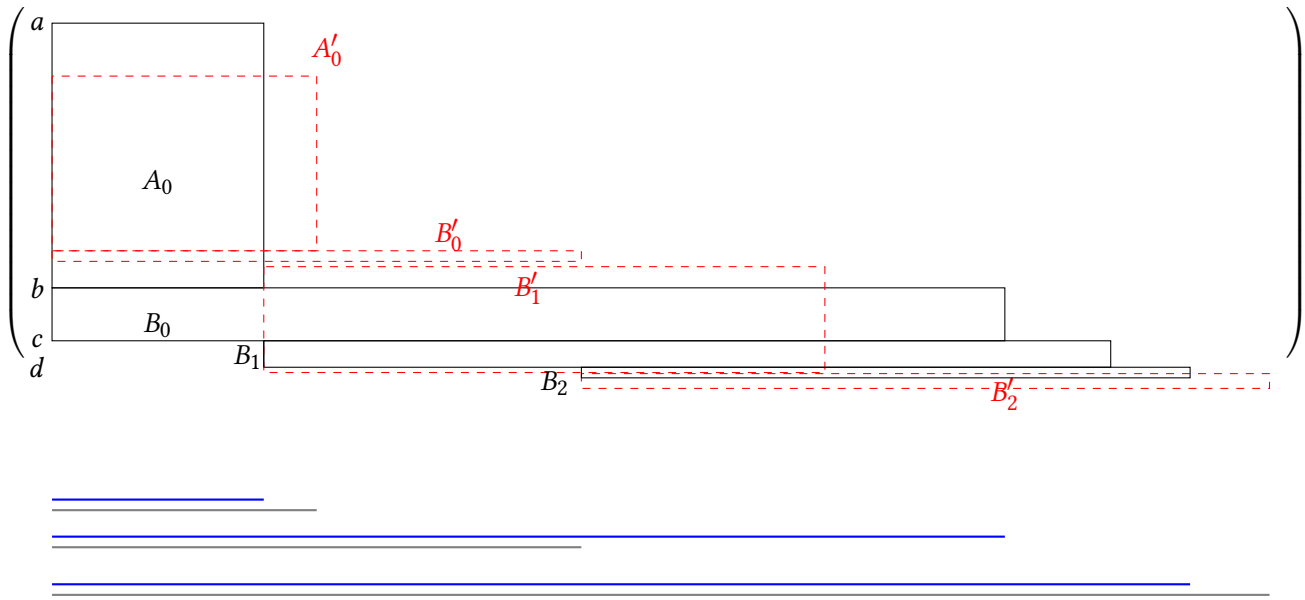


Figure 5: Blocks of an instance M in the DP for the second string. B'_0 is a child of A_0, B'_1 is a child of A_0, B'_1 is a child of A_0. The blue and gray lines represents σ and σ' respectively from first two iterations of DP. The sketch shows the switch example in the second iteration.

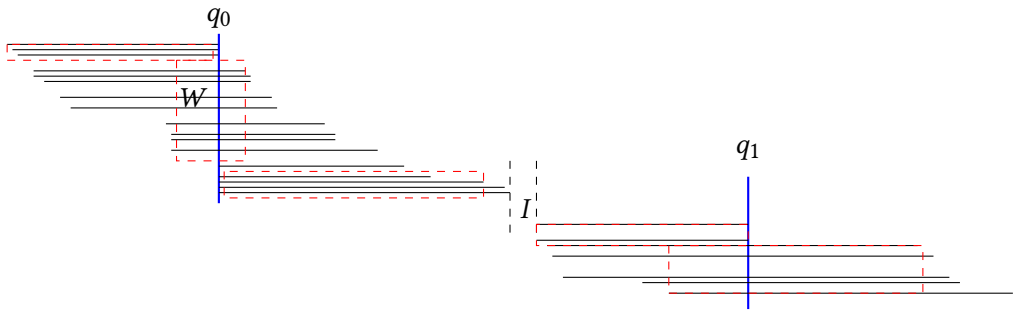


Figure 6: Blocks represented by ranges shown in red on an instance M and the blue lines are the columns, I and C shows the empty interval and central region respectively.

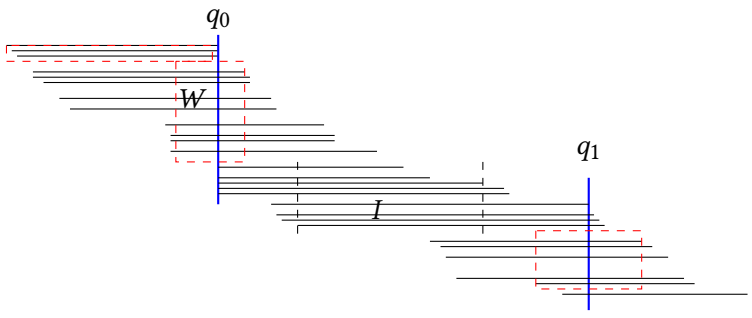


Figure 7: This sketch shows non-dominance example in region I.

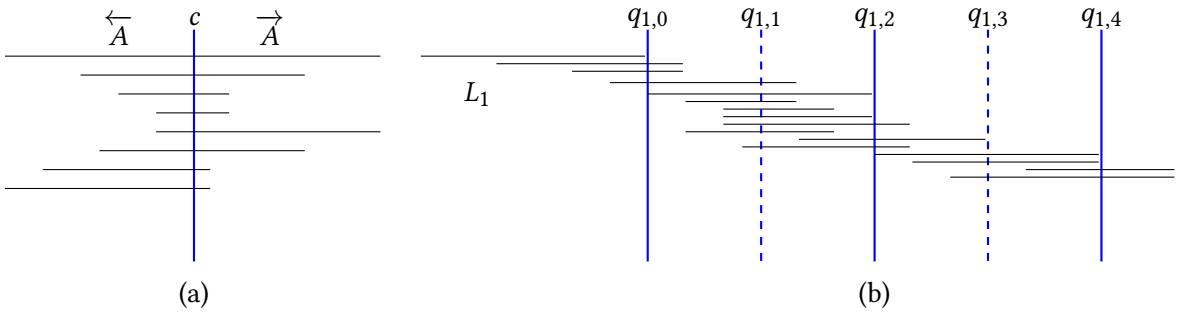


Figure 8: (a) Sub-matrices for rooted Gapless-MEC. (b) For a single-length-class instance, the sketch shows the strings crossing each column either exactly once or exactly twice.

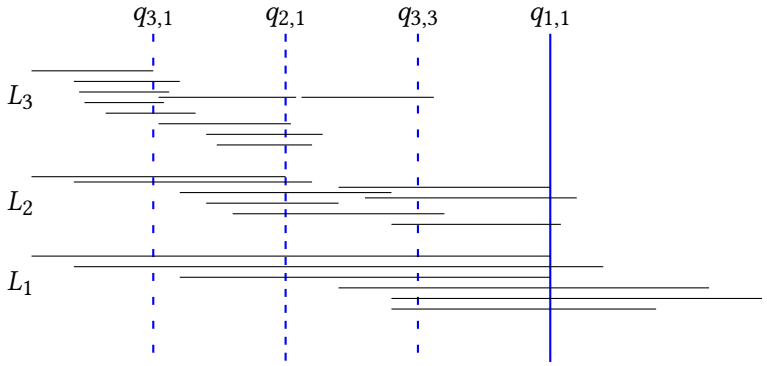


Figure 9: Different length classes L_1 with corresponding column $q_{1,1}$, L_2 with corresponding columns $q_{2,1}$, $q_{2,2} = q_{1,1}$, and L_3 with corresponding columns $q_{3,1}$, $q_{3,2} = q_{2,1}$, $q_{3,3}$, $q_{3,4} = q_{1,1}$.

Chapter **A**

Graph Algorithms

A.1 Introduction

To determine the two genome sequences of dilpoid organisms is important to understand the biology related to diseases. This is required to correctly understand allele-specific expression and compound heterozygosity. We refer the reader to (Tewhey et al., 2011a; Glusman et al., 2014b) for more details. Thanks to next-generation sequencing technologies that have the power to deliver the pieces of these genome sequences called as sequencing redas. The goal is to reconstruct the two underlying genome sequences from these reads. The process of assemblying the two genome sequences from the reads is known as diploid genome assembly. Solving an error-free version of diploid genome assemblies is pretty straightforward. Due to the large amount of data, sequencing errors and genomics repeats, the diploid genome assemblies is a fundamental challenging problem.

Since the era of Sanger and Illumina sequencing, denovo assembly of heterozygous diploid genomes have been a difficult problem Vinson et al. (2005). There are several short-read assemblers using Illumina data for heterozygous genomes Kajitani et al. (2014); Pryszcz and Gabaldón (2016); Safonova et al. (2015); Bankevich et al. (2012). The sequences delivered from these assemblers are short and contain gaps. More recently, some 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) use the long PacBio sequencing data for denovo assembly Most of the assemblers involve collapsing the two genome sequences into single haploid "consensus" sequence. See review paper for literature about denovo assembly approaches (Sović et al., 2013; Myers, 1995, 2005; Nagarajan and Pop, 2009). This is achieved by merging the distinct alleles at regions of heterozygosity into single allele, and therefore, loosing lot of information. Moreover, haploid sequence does not represent the true characteristics of the diploid genome.

On the other hand, there are two standard linear approaches to generate the diploid assemblies, one by using haploid sequences also known as contigs Chin et al. (2016); Pendleton et al. (2015); Seo et al. (2016); Mostovoy et al. (2016) and another by using the reference genome Glusman et al. (2014b). In both the approaches, first, the reads are aligned to the reference genome or contigs; second, we call variants such as SNVs from the short Illumina reads; third, we phase the detected SNVs using reads from different sequencing technologies. To solve phasing problem, it is generally formulated as minimum error correction optimization problem (Cilibrasi et al., 2007c). See review (Rhee et al., 2016) for details. In the recent study by (Chin et al., 2016), these diploid de novo assemblies have been demonstrated for small and mid-sized genomes using PacBio data. On the other hand, (Weisenfeld et al., 2017) et al. provides a scalable method for determining the actual diploid genome sequence in a sample using 10x Genomics microfluidic platform. In both the studies, first, they generate linear contigs from the assembly graph; second, the reads are re-aligned to these contigs to detect the regions of heterozygosity and third, the phasing is performed to get haplotigs.

In the reference guided assembly, the reads are first aligned to the reference genome and therefore contain the reference bias. They fail to see the sequences or variations that are unique to the genome. There are several reasons why the set of sequences/contigs is not ideal: First, the contigs produced by

the assemblers ignore the variants in the complex regions.. Second, the contigs does not capture the whole information of the genome. Third, they are very fragmented and the storage of contigs is highly inefficient in both terms of storage and searching due to high similarity.

Here, we propose an alternative approach to Falcon unzip to use the underlying assembly graph as reference when a fully assembled genome is not available. Our pipeline on diploid assembly can be integrated with any pre-existing denovo assembler. We develop a pipeline to perform haplotype-aware denovo assembly in order to generate diploid assemblies. Potentially, we are interested to generate longer assemblies from the reads that span the whole genome. It is hard to achieve longer diploid assemblies by using high coverage short Illumina reads only that results in more fragmented ones. Therefore, we augment the high quality short read Illumina data with third generation sequencing technologies data like Pacific Biosciences (PacBio) and Oxford Nanopore (ONT). These technologies deliver sequencing reads in the order of thousand to tens of thousands of nucloetides, and therefore, potentially contain more phase information, but their downside is the higher error rate compared to the Illumina data. To this end, we provide an efficient automatic pipeline to obtain accurate and comprehensive phase information directly from the combined Illumina and PacBio readset for a single diploid organism.

Our contribution is three fold: First, we generalise this problem to construct the diploid assemblies directly from the underlying assembly graph. The two genome sequences can be seen as the two optimal paths over the regions of heterozygosity in assembly graph. Second, we propose an hybrid approach to integrate accurate Illumina and long-read PacBio reads to generate diploid assemblies. Illumina reads being accurate are used to generate an assembly graph as backbone for downstream steps. Further, we augment the assembly graph with long PacBio reads, that potentially connect the contigs to get longer assemblies (scaffolds) that helps to resolve more complex regions of the genome. Third, this work is one of the first step in the direction of performing read-based phasing on graphs, although we consider simple setting here.

We demonstrate the feasibility of this approach by performing haplotype-aware de novo genome assembly of a whole pseudo-yeast (SK1+Y12) genome. We show that we generate more accurate, contiguous, and correctly phased diploid genomes compared to linear appraoch. **[TODO: Maybe try HUMAN SAMPLE NA12878 and mention results properly for real data.]**

A.2 Diploid assembly pipeline

In this section, we describe our automatic "push-button" pipeline to obtain diploid assemblies by combining Illumina and PacBio data. **[TODO: add figure to describe all the steps.]**

A.2.1 Generation of assembly graph

We perform error-correction of Illumina reads from diploid organism using FermiKit (?). Then, the reads are denovo assembled to generate an assembly string graph.

Definition A.1 (Overlap Graph). Formally, Let X be a string of symbols $x_1 \dots x_l$ from an alphabet $\mathcal{A} = \{A, C, G, T\}$. The length of string X is denoted by $|X|$. A substring of X is denoted by $X[i, j]$ where $1 \leq i \leq j \leq |X|$. A substring of the form $X[1, i]$ is a prefix of X and a substring $X[k, |X|]$ is a suffix of X. We use $\overline{X}$ to denote the reverse complement of a string. An overlap graph is formed from a set of reads R by finding all pairwise overlaps of length at least τ between members of R. We say that two reads X and Y overlap when a suffix of X matches a prefix of Y or vice versa. Therefore, in the overlap graph $G(V, E)$, $V = \{v_1, \dots v_i\}$ be the set of reads R and $E = \{e_{12}, \dots e_{ij}\}$ be a set of directed edges when a suffix of v_i matches a prefix of v_j.

Furthermore, the overlap graph G can be reduced to the string graph by first removing duplicate reads (distinct elements of R with the same or reverse-complemented sequence) and contained reads (elements in R that are a substring of some element in R or their reverse complements), then removing transitive edges from the graph.

A.2.2 Conversion to sequence graph

We convert the assembly string graph to sequence graph.

Definition A.2 (Sequence Graph). We define a sequence graph $G_s(N_s, E_s)$ as a collection of nodes and egdes, in which each endpoint of every edge has an independent orientation (denoted either "left" or "right"), indicating if the endpoint is incident with the *left* or *right side* of the given vertex. The sides of G_s are therefore the set $N_s \times \{left, right\}$, and each edge in E_s is a pair set of two sides . We say for all $n_i \in N_s$, $(n_i, left)$ and $(n_i, right)$ are *opposite sides*. The nodes n_i are the sequences from an alphabet $\mathcal{A} = \{A, C, G, T\}$. Nodes may be traversed in either the forward or reverse direction, with the sequence being reverse-complemented in the reverse direction. The edge e_{ij} connect sides of the nodes $(n_i, left)$ to $(n_j, right)$ represent adjacencies between the sequences of the nodes they connect. Thus, the graph implicitly encodes longer sequences as the concatenated node sequences along walks through the graph.

We perform *bluntification* process to remove the overlaps between the nodes to generate sequence graph G_s. The bluntification step is implemented using pinch library, which is basically the transitive closure of the overlap relation.

More precisely, if there is a directed edge from node $n_1 = X[1, |X|]$ to node $n_2 = Y[1, |Y|]$ in the string graph G, then the suffix $X[k, |X|]$ overlaps with prefix $Y[1, k]$. When we convert it into sequence graph G_s, the nodes are updated with $n_{s1} = X[1, k]$ and $n_{s2} = Y[k, |Y|]$ in the sequence graph G_s and in addition, a new node is created based on an overlap $n_{s3} = X[k, |X|]$.

A.2.3 Simple bubble detection in sequence graph

To account for heterozygosity in a diploid genome, we perform simple detection. We define a simple bubble that has the following properties:

- *Disjoint paths.* A set of paths that start and end at common source and sink nodes but are otherwise disjoint.
- *Similarity.* The path sequences differ from each other by a single allele base.
- *Directionality.* Consider simple paths that are all left-to-right ot right-to-left.
- *DAG.* Directed and acyclic.

In the context of sequence analysis, a bubble can represent a potential sequencing error or a genetic variation within a set of homologous molecules. Due to the sequencing errors, there could be complicated structures in the bubbles, in particular we focus on simple bubbles that may represent multi-allelic SNP sites.

Formally, we represent ordering of bubbles as *locus* records in the sequence graph .

Definition A.3 (Locus). We define the set of loci $L = \{l_1, l_2, \ldots l_k\}$ as the simple bubbles (represent multi-allelic SNP sites) and its repective directed paths. Formally, one locus record is represeted as

$$l_k = \{p_1, p_2 \ldots\}$$

Definition A.4 (Path). We define path p_k as linear ordering of vertices, Formally,

$$p_z = (n_1, right), \ldots (n_m, left) \ldots$$

A.2.4 PacBio alignments

The next stage of the pipeline is to build the optimal genome paths in the sequence graph G_s that are consistent with the aligned reads. Here, we take advantage of third generation sequencing technologies that are long and have tendency to connect contigs to generate long assemblies. Therefore, we align the long PacBio reads to G_s using Mikko's mapping algorithm (**?**). It is build on the ideas of V-ALIGN, in combination with the generalization of Myer's bit vector algorithm.

Definition A.5 (Alignment). We define read-alignments $R = \{r_1, r_2, \ldots r_h\}$ as a path of vertices through graph G_s such that each vertex contains edit distances. We keep track of each read aligned to forward or reverse strand. Additionally, we keep the aligned read quality scores $q(r)$. Formally, an alignment is represeted as

$$r_h = \{(n_1, left), \ldots (n_m, right) \ldots\}$$

A.2.5 Bubble ordering

The next stage is to get the ordering of the bubbles. This is challenging due to the different type of repeats present in the genome. The tandem repeats creates local cycles and interspersed repeats global cycles in the graph. There could be both inter and intra chromosomal repeats present in the genome. Therefore, the graph is cyclic and to get a linear ordering of bubbles is a challenging problem. Solving the bubble ordering is equivalent to solving the assembly problem. The focus of this study is to provide the diploid assembly with the assumption that we know the ordering of the bubbles. There are many tools such as canu ... (**?**) that can provide the single assembly for highly heterozygous genomes.

A.2.6 Generation of diploid assemblies

[TODO: Phasing repetitive bubbles by unfolding them and filtering the reads that could potentially come from other chromosomes based on the bubbles traversed by a read.] Given the bubble ordering and the PacBio alignments, the goal here is to generate the diploid assemblies for every connected component of alignments. 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. The bubbles in the contigs can be phased by using PacBio reads aligned to them. This process is known as *read-based phasing*.

The read-based phasing is formulated as the Minimum Error Correction (MEC) problem (**?**) and its weighted version (wMEC). Previously, wMEC is solved efficiently for bi-allelic variants (**?**), but here we now generalise it for simple bubbles. Solving phasing problem for simple bubbles is equivalent to phasing the multi-allelic SNP sites.

[TODO: add figure for wMEC on simple bubbles.]

wMEC for simple bubbles. First, we generate an association between simple bubbles L and aligned PacBio reads R in sequence graph G_s in order to re-discover the bubbles in the aligned read. Additionally, we store the information about the path p_i in a bubble l_j that the read r_k is aligned. We represent this association in the form of a SNP matrix $\mathcal{F} \in \{0, 1, \ldots a, -\}^{|R| \times |L|}$, where $|R|$ is the number of reads, $|L|$ is the number of bubbles along a chromosome, $a = |l_k|$ are the number of paths (or alleles) in a bubble l_k. Please note that the different paths in a bubble corresponds to possible alleles at a genetic variant. The entry $e_i \in \{0, 1, \ldots a, -\}$ in $\mathcal{F}(j.k)$ represents the path in a bubble l_k that a read r_j is aligned. The "–" encodes missing information or internal segment of paired-end Illumina reads.

Definition A.6 (Distance). The quality of a solution relies on the measure $d(r_1, r_2)$ based on the Hamming distance between any two rows $r_1, r_2 \in \{0, 1, \ldots a, -\}^M$ in $\mathcal{F}$. Formally,

$$d(r_1, r_2) := \sum_{k=1}^{|L|} \left| \{ k \mid r_1(k) \neq - \ \wedge \ r_2(k) \neq - \ \wedge \ r_1(k) \neq r_2(k) \} \right|.$$

Definition A.7 (Feasibility). A feasible solution to a SNP matrix $\mathcal{F} \in \{0, 1, \ldots a, -\}^{|R| \times |L|}$ is a pair of haplotypes $h^0, h^1 \in \{0, 1, \ldots a\}^M$ such that

$$d(h_0, h_1) := \sum_{j=1}^{|R|} \min\{d(\mathcal{F}(j), h_0), d(\mathcal{F}(j), h_1)\}$$

and 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.

Furthermore, we have "phred-scaled" quality score tuple for each entry in $\mathcal{F}$ that is stored in $\mathcal{W}(j,k) = \{0, q_1, \ldots, q_a\}$. The size of each tuple $|\mathcal{W}(j,k)|$ is equal to number of paths a at bubble l_k. The quality score value "0" in tuple $\mathcal{W}(j,k)$ encodes that the aligned reads r_j to the same path index p_i in a bubble l_k. The remaining non-zero values stores the confidence scores of switching the aligned read r_j to other branches/paths $\{p_i\}$ in a bubble. These phred scores can hence serve as costs of flipping a letter, allowing less confident base calls to be corrected at lower cost compared to high confidence ones.

Problem A.1 (wMEC for simple bubbles). Given a matrix $\mathcal{F} \in \{0, 1, \ldots a, -\}^{|R|\times|L|}$ and a weight matrix $\mathcal{W} \in \mathbb{N}^{|R|\times|L|}$, where each entry in $\mathcal{W}(j,k)$ is a tuple, thereby flipping entries in $\mathcal{F}$ to i^{th} allele in order to obtain a feasible matrix, while minimizing the sum of incurred costs, where flipping entry $\mathcal{F}(j,k)$ incurs a corresponding cost from i^{th} element of tuple $\mathcal{W}(j,k)$. For simplicity, we represent the i^{th} element of tuple $\mathcal{W}(j,k)$ as $\mathcal{W}(j,k,i)$

Therefore, wMEC for bi-allelic is a special case of multi-allelic with $a = \{0, 1\}$.

Algorithm. In WhatsHap algorithm (**?**), wMEC is solved in an exact manner for bi-allelic variants using dynamic programming approach. It runs in $\mathcal{O}(2^c \cdot |L|)$ time, where $|L|$ is the number of variants to be phased and c is the maximum physical coverage (which includes internal segments of paired-end 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 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 A.8 (Bipartition extension). For a given set A and a subset $A' \subset A$, a bipartition $B = (P, Q)$ of A is said to *extend* a bipartition $B' = (P', Q')$ of A' if $P' \subset P$ and $Q' \subset Q$, denoted by $B \simeq B'$.

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

Solving wMEC for simple bubbles. The basic idea is to now extend the dynamic program to consider all possible path-pairs from a bubble. In the bi-allelic case, we have only two paths in every bubble, therefore, the possible path-pairs is one. In the multi-allelic case, we consider all possible path-pairs in a variant/bubble. The goal is to find two optimal minimum cost paths from the simplified version of sequence graph G_s based on bubble-pair phasing. In the algorithm, we move from left to right in DAG connected component, we find the two optimal paths of the first m bubbles and then extend it to find it for $m+1$ bubbles, resulting optimal paths for first $m+1$ bubbles. At every step, we remember the set of haplotype pairs possible from previous step only. In this way, we do this till the last bubble $|L|$ and backtrace to get the two optimal paths. Now, we describe how to build the DP table.

DP cell initialization. Along the similar lines of (**?**), we compute $\Delta_C(k,B)$ for any column k and bipartition B. We write $\mathcal{S}(k,B)$ to denote this set of sets of reads induced by bipartition B in column k. The cost $W^i_{k,S}$ of flipping all entries in a read set $S \in \mathcal{S}(k,B)$ to the some allele $i \in \{0, 1, \ldots a\}$ is given by

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

Now, we minimize the cost by considering all possibilities of pairs of alleles from a bubble. Formally, if we have $|a|$ as the number of possible alleles in a bubble, then the possibilities of heterozygous alleles are $\binom{a}{2}$. The cost $\Delta_C(k,B)$ is computed using Algroithm 3 as shown below:

DP column initialization. Next, we initialize first DP column by $C(1,B) := \Delta_C(1,B)$ for all possible bipartitions B over trying all possible allele-pairs. Next we enumerate all bipartitions in Gray code order, as done previously (**?**). This ensures that only one read is moved from one set to another in each step, facilitating constant time updates of the values $W^i_{k,S}$.

Due to the "algorithm engineering" using Grey code, we can perform this operation for one DP column takes $\mathcal{O}(\binom{a}{2} \cdot 2^{|A(k)|})$ time.

Input : The column vectors of the SNP matrix $\mathcal{F}(k)$ and weight matrix $\mathcal{W}(k)$ of the bubble k and the bipartition S of active reads $R(k)$
Output: $\Delta_C(k, B)$

for *all allele-pairs* $(a_i a_j)$ *from paths a in bubble* k **do**

$$\Delta_C(k, B) = \min_{i \in \binom{a}{2}^{\mathcal{S}(k,B)}} \left\{ \sum_{S \in \mathcal{S}(k,B)} W^i_{k,S} \right\},$$

Algorithm 3: DP CELL INITIALIZATION

Input : $\Delta_C(k, B)$ for all bipartitions of bubble k.
Output: $C(1, B)$

for *all bipartitions* B *of column* k **do**
  Compute $\Delta_C(k, B)$ using Algorithm 3 and store in $C(1, B)$.

Algorithm 4: DP COLUMN INITIALIZATION

DP column recurrence. Note that $C(k, B)$ 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 at bubble k. Entries in column $C(k + 1, \cdot)$ should hence add up local costs incurred in column $k + 1$ and costs from the previous column. To adhere to the semantics of $C(k + 1, B)$, only entries in column k whose bipartitions are *compatible* with B are to be considered as possible "predecessors" of $C(k + 1, B)$.

Input : $C(1, B)$ for all bipartitions of bubble k.
Output: $C(k, B)$ for columns k till last column $|L|$

for *all columns* $k \in \{2 \dots |L|\}$ **do**
  Compute $\Delta_C(k + 1, B)$ using Algorithm 3.
  Consider cost from the previous column computed using Algorithm 4.
  Compute for recurrence cost for column $k + 1$

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

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

Algorithm 5: DP TABLE

Algorithm Engineering. To ease computing $C(k + 1, B)$ in Algorithm 5, we use the same technique described by and define intermediate *projection columns* $C^{\cap}(k, \cdot)$. 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 $2^{|A(k) \cap A(k+1)|}$ entries, which are given by

$$C^{\cap}(k, B') = \min_{\mathcal{B}(A(k)) : B \simeq B'} \{C(k, B)\}. \tag{A.1}$$

These projection columns can be created while computing $C(k, \cdot)$ at no extra (asymptotic) runtime. Using these projection columns, Equation (B.2) becomes

$$C(k + 1, B) = \Delta_C(k + 1, B) + \min C^{\cap}(k, B \cap A(k)),$$

where $B \cap A(k) := (P \cap A(k), Q \cap A(k))$ for $B = (P, Q)$.

Running time. To compute DP column takes time $\mathcal{O}(\binom{a}{2} \cdot 2^{|A(k)|})$ and in total, the running time is $\mathcal{O}(\binom{a}{2} \cdot 2^{|A(k)|} \cdot |L|)$ for $|L|$ bubbles. It is independent of read-length, and therefore, suitable for longer reads from upcoming technologies.

Backtracing. We can backtrace from the last bubble $C(|L|, \cdot)$ to compute haplotypes and optimal read bipartition $B_k^* = (P_k^*, Q_k^*)$ at each bubble k. The two haplotypes (h_0, h_1) can be derived by the following equation

$$h_0(k) = i \quad if \min_{i \in \{0,1,\dots a\}} W^i_{k,P*},$$

$$h_1(k) = j \quad if \min_{j \in \{0,1,\dots a\}} W^j_{k,Q*},$$

A.2.7 Generation of final assemblies

Here, we consider the base sequence graph G_s and (h_0, h_1) to generate final assemblies for every connected component. For every connected component, it is trivial to construct the contig sequences by traversing the haplotype paths and considering the node sequences and its sides information stored in the based graph G_s.

A.3 Dataset

A.3.1 The yeast pseudo-diploid genome

A.3.2 The human genome

A.4 Assembly performance assessment

To access the performace of our pipeline, we compared assemblies of yeast genome using our pipeline to FermiKit (**?**) and Falcon unzip (**?**).

Assembly contiguity. We assessed the contiguity of the assemblies by calculating the contig alignment length N50. By analyzing the contig alignment lengths, as opposed to the length of contigs themselves, we account for misassembled contigs that can inflate the assembly statistics.

Assembly completeness. The contigs assembled by our pipeline covered ?% of the reference genome. We also argue from where the assemblies come in the reference gemome, in addition, where the ends of your "haplotigs" are.

Phasing errror rate. Over the yeast genome, we compare diploid assemblies with "TRUTH" haploid assemblies.

Read partitioning accuracy. Over the yeast genome, we compare reads that the haplotype they belongs to the "TRUTH" partitioning of the reads.

A.5 Result

[TODO: Focus only on the haplotigs.]

A.5.1 Coverage Analysis of PacBio reads

We downsample the PacBio data to different coverages (2x, 3x, 4x, 5x, 10x, 15x, 30x). Table shows the following stats for both linear and graph version:

1. Assembly size
2. Contigs
3. N50 size
4. N50 no.
5. N90 size
6. Max contig size
7. percent identity

The plot shows the phasing error rate for both linear and graph version for different coverages.

Chapter **B**

Parameterized algorithm for pedigree-based phasing

Generalization to Pedigrees

Motivation: 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.
Results: 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.
Availability: https://bitbucket.org/whatshap/whatshap
Contact: t.marschall@mpi-inf.mpg.de

B.1 Introduction

Humans are diploid and determining the sequences of both homologous copies of each chromosome is desirable for many reasons. These two sequences per chromosome are known as *haplotypes*. The process of obtaining them is known as *haplotyping* or *phasing*, two terms that we will use interchangeably. Haplotype-resolved genetic data can be used, for instance, for population genetic analyses of admixture, migration, and selection, but also to study allele-specific gene regulation, compound heterozygosity, and their roles in human disease. We refer the reader to Tewhey et al. (2011b) and Glusman et al. (2014c) for detailed reviews on the relevance of haplotyping.

General Approaches for Haplotyping. There are three major approaches to phasing. 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). This approach is known as *statistical* or *population-based phasing*. 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., 2014c). 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 phasings (Roach et al., 2011a). 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 largely differ in the amount of work, DNA, and money they require. 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. Another approach consists in breaking both homologous chromosomes into (larger) fragments and separating them into a number of pools such that each pool is unlikely to contain fragments from the same locus of both haplotypes. This can, for instance, be achieved by dilution followed by bar-coded short-read sequencing. To achieve molecular haplotyping over the range of a full chromosome, protocols have been invented to physically separate the two homologous chromosomes, for example by microscopy-based chromosome isolation, fluorescence-activated sorting, or microfluidics-based sorting. These and other experimental techniques for molecular haplotyping have been surveyed by Snyder et al. (2015b). They are of great interest because they facilitate phasing of rare variants for single individuals. Rare variants have been postulated to contribute considerably to clinical traits and are hence of major interest.

Haplotype Assembly. When many haplotype fragments are available for one individual, for instance from sequencing, one can attempt to reconstruct the full haplotypes or at least to obtain larger blocks. This process is known as *haplotype assembly*, *single-individual haplotyping*, or *read-based phasing* (in case the fragments indeed stem from sequencing reads). It requires reads that span two or more heterozygous variants. In order to be successful, reads covering as many pairs of consecutive heterozygous variants as possible are desirable. At present, third generation sequencing platforms, as marketed by Pacific Biosciences (PacBio) and Oxford Nanopore, become more widespread and offer reads spanning thousands to tens of thousands of nucleotides. Although error-rates are much higher than for common second generation technologies, the longer reads provide substantially more phase information and hence render them promising platforms for read-based phasing.

We adapt the common assumption that all variants to be phased are bi-allelic and non-overlapping. Then, the input to the haplotype assembly problem can be specified by a matrix with entries from $\{0, 1, -\}$, having one row per read and one column per variant. Entries of 0 or 1 indicate that the read in that row gives evidence for the reference or alternative allele for the variant in that column, respectively. An entry of "–" signifies that a variant is not covered by that read. In case all reads are error-free and mapped to correct positions, the set of rows in that matrix admits a bipartition such that each of the two partitions is *conflict free*. Here, two rows are defined to be conflicting if they exhibit different non-dash values in the same column; that is, one entry is 0 and the other one is 1. If such a bipartition exists, the matrix is called *feasible*.

To formalize the haplotype assembly problem in the face of errors, we define *operations* on the input matrix and ask for the minimum number of operations one needs to apply to render it feasible. Different such operations have been studied, in particular removal of rows, resulting in the *Minimum Fragment Removal (MFR)* problem, removal of columns, resulting in the *Minimum SNP Removal (MSR)* problem, and flipping of bits, resulting in the *Minimum Error Correction (MEC)* problem. All three problems are NP-hard (Lancia et al., 2001b; Cilibrasi et al., 2007a). Flipping of bits corresponds to correcting sequencing errors and hence the MEC problem has received most attention in the literature and is most relevant in practice. A wealth of exact and heuristic approaches to solve the MEC problem exists. Exact approaches, which solve the problem optimally, include integer linear programming (Fouilhoux and Mahjoub, 2012a; Chen et al., 2013c), and fixed-parameter tractable (FPT) algorithms (He et al., 2010a; Patterson et al., 2015a; Pirola et al., 2015a). Refer to the reviews by Schwartz (2010) and Rhee et al. (2015) for further related approaches.

Here, we build upon our previous approach WhatsHap (Patterson et al., 2014, 2015a) and generalize it to jointly handle sequencing reads of related individuals. WhatsHap is an FPT approach that solves

the (weighted) MEC problem optimally in time exponential in the maximum coverage, but linear in the number of variants. In particular the runtime does not explicitly depend on the read length. These properties make it particularly apt for current long-read data. This has also been observed by Kuleshov (2014), who approached the weighted MEC problem in a message-passing framework and, by doing so, independently arrived at the same DP algorithm used in WhatsHap. The exponential runtime in the maximum coverage does not constitute a problem in practice because reads can be removed in regions of excess coverage without loosing much information. The evaluation by Patterson et al. (2015a) suggests that pruning data to a maximum coverage of 15× yields excellent results while an even higher coverage does not deliver a big additional improvement.

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. (2013), 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., 2013a). 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 introduce a unifying formal framework to fully integrate read-based and genetic haplotyping. 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 runtime 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 microarrays or short-read sequencing. On simulated and real PacBio data, we show that sequencing each individual in a mother-father-child trio to 5× 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× 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.

Example. Figure B.1 shows seven SNP positions covered by reads in three related individuals. It 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. By observing that all seven child genotypes are compatible with the combination of brown and green haplotypes from the parents, however, these phases can be easily inferred. This example demonstrates that jointly using pedigree information, genotypes, and sequencing reads is very powerful for establishing phase information.

Figure B.1: 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.

B.2 The Weighted Minimum Error Correction Problem on Pedigrees

Thus far, read-based phasing has predominantly been formulated as the Minimum Error Correction (MEC) problem (Cilibrasi et al., 2007a) and its weighted sibling wMEC (Greenberg et al., 2004). We first re-state these problems formally to introduce our notation and then proceed to generalizing them to pedigrees.

The input to the MEC problem consists of a SNP matrix $\mathcal{F} \in \{0, 1, -\}^{R \times M}$, 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.

Definition B.1 (Distance). Given two vectors $r_1, r_2 \in \{0, 1, -\}^M$, the *distance* $d(r_1, r_2)$ between r_1 and r_2 is given by the number of mismatching non-dash characters. Formally,

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

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

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

The MEC problem is NP-hard (Cilibrasi et al., 2007a). The weighted version of the problem associates a cost to every matrix entry. This is useful 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 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 lower cost compared to high confidence ones.

Problem B.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 $\mathcal{W}(j, k)$.

Table B.1: Overview of common notation.

Notation	Meaning	Example
$\mathcal{I}$	Set of individuals	$\{1, 2, 3, 4\}$
$\mathcal{T}$	Set of trio relationships	$\{(1, 2, 3), ($
$\mathcal{F}_i \in \{0, 1, -\}^{R_i \times M}$	Input SNP matrix for individual $i \in \mathcal{I}$	$\begin{bmatrix} - & - \\ 0 & 1 \end{bmatrix}$
$\mathcal{W}_i \in \mathbb{N}^{R_i \times M}$	Matrix of weights for individual $i \in \mathcal{I}$	$\begin{bmatrix} 0 & 0 \\ 13 & 9 \end{bmatrix}$
$\mathcal{X} \in \mathbb{N}^M$	Recombination cost vector	$(5, 20, 12,$
$g_i \in \{0, 1, 2\}^M$	Input genotypes for individual i	$(0, 2, 2, 1,$
$A(k)$	Set of reads active in column k	$\{[- - 1 0$
$\Delta_C(k, B, t)$	Local cost incurred 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	$(0, 1, 1, 1,$
$t_{m\to c}, t_{f\to c} \in \{0, 1\}^M$	Sought transmission vectors for trio $(m, f, c) \in \mathcal{T}$	$(0, 0, 0, 1,$

We now generalize wMEC to account for multiple individuals in a pedigree simultaneously while modeling inheritance and recombination. An overview of notation we use is provided in Table B.1. We assume our pedigree to contain a set of N individuals $\mathcal{I} = \{1, \ldots, N\}$. 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 B.3 (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 B.4 (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 B.3 (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 B.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 B.4 (PedMEC with genotypes, PedMEC-G). Let the same input be given as for Problem B.3 (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., 2013c). 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.

B.3 Algorithm

Solving MEC and wMEC. WhatsHap (Patterson et al., 2015a) is a dynamic programming (DP) algorithm to optimally solve the wMEC problem. 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). 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 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 B.5 (Bipartition extension). For a given set A and a subset $A' \subset A$, a bipartition $B = (P, Q)$ of A is said to *extend* 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.

Algorithm Overview: Solving PedMEC and PedMEC-G. In the following, we will see how this idea 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. 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

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

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. (2015a), 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

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

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., 2015a). 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 (B.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 B.6 (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

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

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 (B.2), we use the same technique described by Patterson et al. (2015a) 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

$$C^{\cap}(k, B', t) = \min_{\mathcal{B}(A(k)):B\simeq B'} \{C(k, B, t)\}. \tag{B.3}$$

These projection columns can be created while computing $C(k, \cdot, \cdot)$ at no extra (asymptotic) runtime. Using these projection columns, Equation (B.2) becomes

$$\begin{aligned} C(k+1, B, t) &= \Delta_C(k+1, B, t) \\ &+ \min_{t'\in\{0,1\}^{2|\mathcal{T}|}} \left\{C^{\cap}(k, B\cap A(k), t') + d_H(t, t')\cdot \mathcal{X}(k+1)\right\}, \end{aligned} \tag{B.4}$$

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

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

B.4 Experimental Setup

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

B.4.1 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[1], the corresponding genetic map[2], 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.

[1] https://mathgen.stats.ox.ac.uk/impute/1000GP_Phase3.tgz
[2] http://www.shapeit.fr/files/genetic_map_b37.tar.gz

PacBio Data. For each individual, we downloaded aligned Pacific Biosciences (PacBio) reads[3], which had an average coverage of 42.3× in mother, 46.8× in father and 60.2× in child, respectively. The average mapped read length across mother was 8,328 bp, father was 8,471 bp and child was 8,687 bp. 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[4] for comparison purposes.

B.4.2 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 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.

B.4.3 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 B.3 have been implemented in the WhatsHap software package[5], distributed as Open Source software under the terms of the MIT license. We emphasize that WhatsHap solves wMEC and PedMEC-G optimally. Since the focus of this paper is on comparing these two models,

[3] 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/

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

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

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 runtime 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 B.3), 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.

B.4.4 Performance Metrics

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

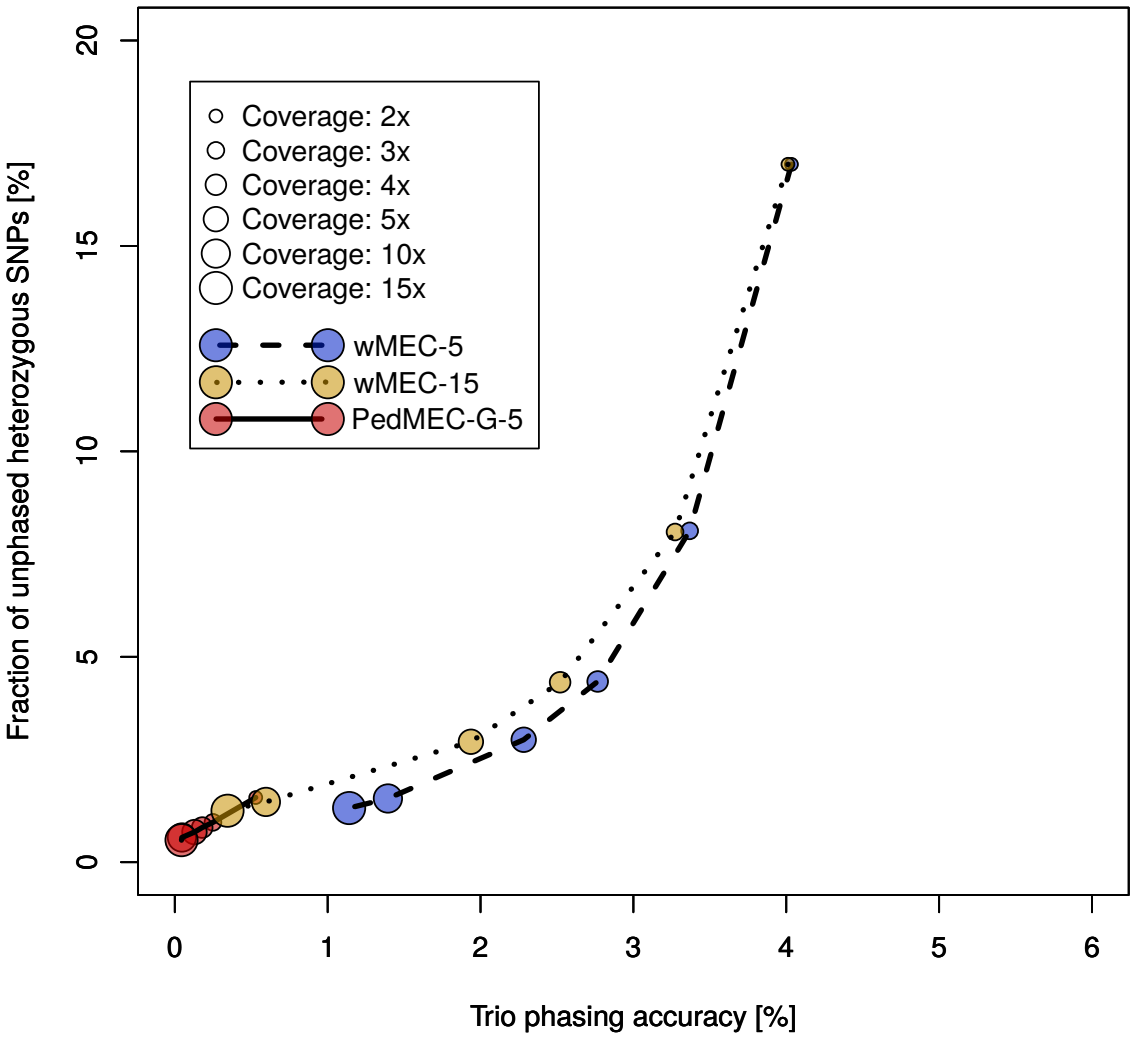
Phased SNPs. For read-based phasing of a single individual (wMEC), we say that two heterozygous SNPs are directly connected if there exists a read covering both. We compute the connected components in the graph where SNPs are nodes and edges are drawn between directly connected SNPs. Each connected component 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 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.

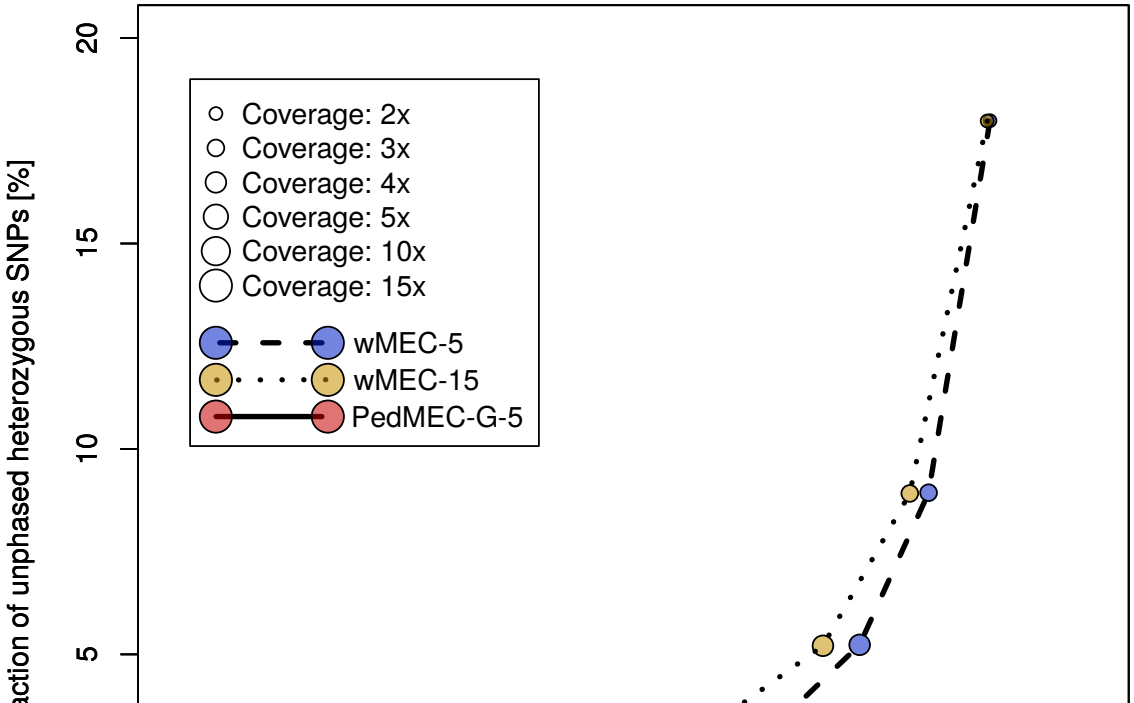
Phasing Error Rate. For each block, the first predicted haplotype is expressed as a mosaic of the two true haplotypes, minimizing the number of switches. This minimum is known as the *number of switch errors*. Note that the second predicted haplotype is exactly the complement of the first one, due to 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.

B.5 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×, 4×, 5×, 10× and 15×) were run. The predicted phasings are compared to the ground truth phasing


Coverage: 2x
Coverage: 3x
Coverage: 4x
Coverage: 5x
Coverage: 10x
Coverage: 15x
wMEC-5
wMEC-15
PedMEC-G-5
Fraction of unphased heterozygous SNPs [%]
Trio phasing accuracy [%]
0
5
10
15
20
0
1
2
3
4
5
6


Coverage: 2x
Coverage: 3x
Coverage: 4x
Coverage: 5x
Coverage: 10x
Coverage: 15x
wMEC-5
wMEC-15
PedMEC-G-5
0
5
10
15
20

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 B.2 shows the fraction of unphased SNPs in comparison to the phasing error rate (see Section B.4.4) 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 B.2. 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 B.2, 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 B.3(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 B.3(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 B.4 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.

B.6 Discussion

We have presented a unifying framework for integrated read-based and genetic haplotyping. By generalizing the WhatsHap algorithm (Patterson et al., 2015a), 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 runtime 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 runtime 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 B.1 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× per individual, on par with performance delivered by single-individual haplotyping at 15× coverage per individual.

Future work. We plan to implement phasing of *de novo* variants observed in the child, which would be impossible with pure genetic haplotyping but is straightforward with our approach.

Since runtime is exponential in the maximum physical coverage, pruning of datasets is required in practice. The read selection approach currently implemented in WhatsHap (Fischer and Marschall, 2016) aims to retain reads that both cover and connect many variants at the same time, in particular for heterogeneous combinations of datasets such as paired-end or mate-pair reads together with long reads. For pedigrees, each dataset is currently pruned individually, but results would likely improve if pedigree structure was taken into account in this step. Finally, since we show that it is possible and beneficial to integrate both read-based and genetic phasing, the next obvious question is whether it is possible to modify our unified theoretical framework to one that also includes statistical phasing.

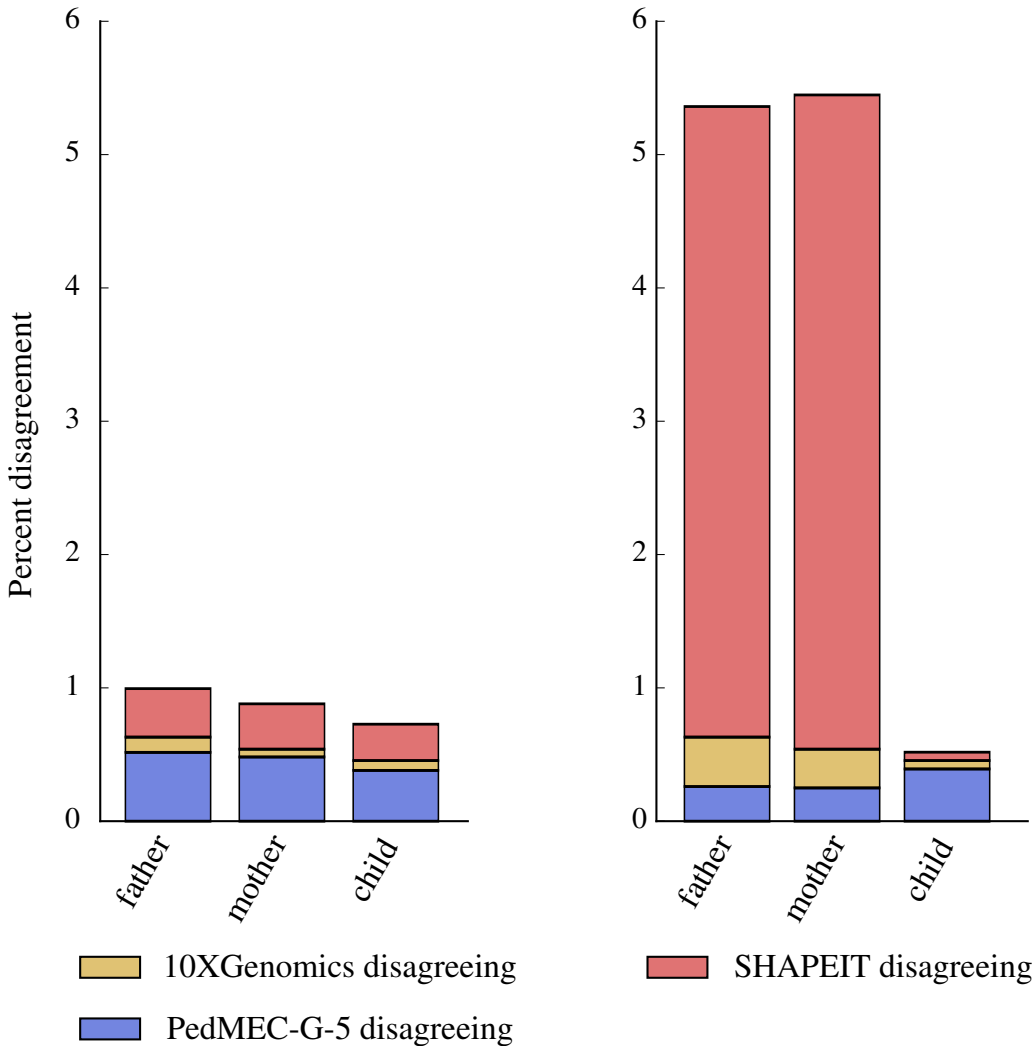


Figure B.3: Three-way comparison of phasings provided by SHAPEIT, 10XGenomics, and PedMEC-G-5 (on 15× coverage data). Of all pairs of consecutive SNPs phased by all three methods, the percentages of cases where the phasing reported by one method disagrees with the other two are reported. Missing to 100%: cases where all three methods agree. Left: SHAPEIT run with default parameters, corresponding to our "ground truth phasing"; right: SHAPEIT run with pedigree information.

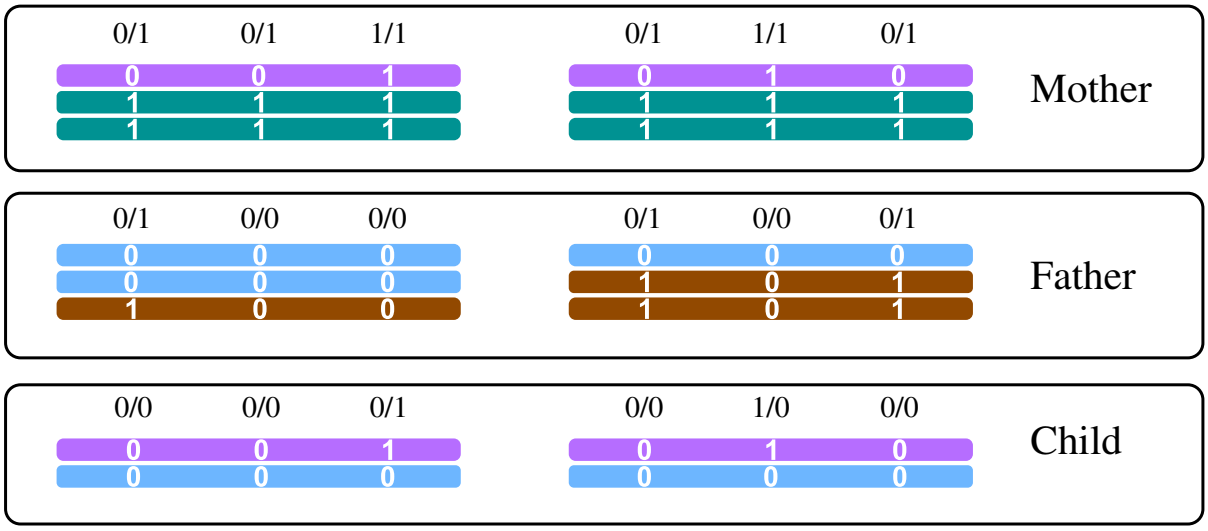


Figure B.4: Two disjoint unconnected haplotype blocks for which phase information can be inferred from the genotypes.

Chapter **C**

Conclusions

Bibliography

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

Aird, D., Ross, M. G., Chen, W.-S., Danielsson, M., Fennell, T., Russ, C., Jaffe, D. B., Nusbaum, C., and Gnirke, A. (2011). Analyzing and minimizing pcr amplification bias in illumina sequencing libraries. *Genome biology*, 12(2):R18.

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

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.

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):623–630.

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*.

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

Chaisson, M. J., Huddleston, J., Dennis, M. Y., Sudmant, P. H., Malig, M., Hormozdiari, F., Antonacci, F., Surti, U., Sandstrom, R., Boitano, M., et al. (2015). Resolving the complexity of the human genome using single-molecule sequencing. *Nature*, 517(7536):608–611.

Chen, W., Li, B., Zeng, Z., Sanna, S., Sidore, C., Busonero, F., Kang, H. M., Li, Y., and Abecasis, G. R. (2013a). Genotype calling and haplotyping in parent-offspring trios. *Genome Research*, 23(1):142–151.

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

Chen, Z.-Z., Deng, F., and Wang, L. (2013c). 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.

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

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

Cilibrasi, R., Van Iersel, L., Kelk, S., and Tromp, J. (2007c). The complexity of the single individual snp haplotyping problem. *Algorithmica*, 49(1):13–36.

Clarke, J., Wu, H.-C., Jayasinghe, L., Patel, A., Reid, S., and Bayley, H. (2009). Continuous base identification for single-molecule nanopore dna sequencing. *Nature nanotechnology*, 4(4):265–270.

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

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

Delaneau, O., Howie, B., Cox, A. J., Zagury, J.-F., and Marchini, J. (2013). 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.

Dohm, J. C., Lottaz, C., Borodina, T., and Himmelbauer, H. (2008). Substantial biases in ultra-short read data sets from high-throughput dna sequencing. *Nucleic acids research*, 36(16):e105–e105.

Dondi, R. (2012). New results for the longest haplotype reconstruction problem. *Discrete Applied Mathematics*, 160(9):1299–1310.

Drmanac, R., Sparks, A. B., Callow, M. J., Halpern, A. L., Burns, N. L., Kermani, B. G., Carnevali, P., Nazarenko, I., Nilsen, G. B., Yeung, G., et al. (2010). Human genome sequencing using unchained base reads on self-assembling dna nanoarrays. *Science*, 327(5961):78–81.

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.

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

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

Forgetta, V., Leveque, G., Dias, J., Grove, D., Lyons Jr, R., Genik, S., Wright, C., Singh, S., Peterson, N., Zianni, M., et al. (2013). Sequencing of the dutch elm disease fungus genome using the roche/454 gs-flx titanium system in a comparison of multiple genomics core facilities. *Journal of biomolecular techniques: JBT*, 24(1):39.

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

Fouilhoux, P. and Mahjoub, A. R. (2012b). 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.

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

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

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

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.

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. (2010a). Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 26(12):i183–i190.

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

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

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.

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.

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. (2001a). 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.

Lancia, G., Bafna, V., Istrail, S., Lippert, R., and Schwartz, R. (2001b). 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.

Leung, D., Jung, I., Rajagopal, N., Schmitt, A., Selvaraj, S., Lee, A. Y., Yen, C.-A., Lin, S., Lin, Y., Qiu, Y., et al. (2015). Integrative analysis of haplotype-resolved epigenomes across human tissues. *Nature*, 518(7539):350–354.

Levene, M. J., Korlach, J., Turner, S. W., Foquet, M., Craighead, H. G., and Webb, W. W. (2003). Zero-mode waveguides for single-molecule analysis at high concentrations. *Science*, 299(5607):682–686.

Li, H. (2013). Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint arXiv:1303.3997*.

Li, M., Ma, B., and Wang, L. (2002). Finding similar regions in many sequences. *J. Comput. Syst. Sci.*, 65(1):73–96.

Li, Z., Zhou, W., Zhang, X.-S., and Chen, L. (2005). A parsimonious tree-grow method for haplotype inference. *Bioinformatics*, 21(17):3475–3481.

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.

Liu, L., Li, Y., Li, S., Hu, N., He, Y., Pong, R., Lin, D., Lu, L., and Law, M. (2012). Comparison of next-generation sequencing systems. *BioMed Research International*, 2012.

Loman, N. J., Misra, R. V., Dallman, T. J., Constantinidou, C., Gharbia, S. E., Wain, J., and Pallen, M. J. (2012). Performance comparison of benchtop high-throughput sequencing platforms. *Nature biotechnology*, 30(5):434–439.

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.

Margulies, M., Egholm, M., Altman, W. E., Attiya, S., Bader, J. S., Bemben, L. A., Berka, J., Braverman, M. S., Chen, Y.-J., Chen, Z., et al. (2005). Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, 437(7057):376–380.

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.

McCarroll, S. A. and Altshuler, D. M. (2007). Copy-number variation and association studies of human disease. *Nature genetics*, 39:S37–S42.

McCoy, R. C., Taylor, R. W., Blauwkamp, T. A., Kelley, J. L., Kertesz, M., Pushkarev, D., Petrov, D. A., and Fiston-Lavier, A.-S. (2014). Illumina truseq synthetic long-reads empower de novo assembly and resolve complex, highly-repetitive transposable elements. *PloS one*, 9(9):e106689.

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.

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.

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.

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. (2015a). WhatsHap: Weighted haplotype assembly for future-generation sequencing reads. *Journal of Computational Biology*, 22(6):498–509.

Patterson, M., Marschall, T., Pisanti, N., van Iersel, L., Stougie, L., Klau, G. W., and Schönhuth, A. (2015b). 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. (2015a). HapCol: accurate and memory-efficient haplotype assembly from long reads. *Bioinformatics*, page btv495.

Pirola, Y., Zaccaria, S., Dondi, R., Klau, G. W., Pisanti, N., and Bonizzoni, P. (2015b). HapCol: accurate and memory-efficient haplotype assembly from long reads. *Bioinformatics*, page btv495.

Pryszcz, L. P. and Gabaldón, T. (2016). Redundans: an assembly pipeline for highly heterozygous genomes. *Nucleic acids research*, 44(12):e113–e113.

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. (2015). Survey of computational haplotype determination methods for single individual. *Genes & Genomics*, pages 1–12.

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.

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. (2011a). Chromosomal haplotypes by genetic phasing of human families. *The American Journal of Human Genetics*, 89(3):382–397.

Roach, J. C., Glusman, G., Hubley, R., Montsaroff, S. Z., Holloway, A. K., Mauldin, D. E., Srivastava, D., Garg, V., Pollard, K. S., Galas, D. J., et al. (2011b). Chromosomal haplotypes by genetic phasing of human families. *The American Journal of Human Genetics*, 89(3):382–397.

Rothberg, J. M., Hinz, W., Rearick, T. M., Schultz, J., Mileski, W., Davey, M., Leamon, J. H., Johnson, K., Milgrew, M. J., Edwards, M., et al. (2011). An integrated semiconductor device enabling non-optical genome sequencing. *Nature*, 475(7356):348–352.

Safonova, Y., Bankevich, A., and Pevzner, P. A. (2015). dipspades: assembler for highly polymorphic diploid genomes. *Journal of Computational Biology*, 22(6):528–545.

Schwartz, R. (2010). Theory and algorithms for the haplotype assembly problem. *Communications in Information & Systems*, 10(1):23–38.

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.

Snyder, M. W., Adey, A., Kitzman, J. O., and Shendure, J. (2015a). Haplotype-resolved genome sequencing: experimental methods and applications. *Nature Reviews Genetics*, 16(6):344–358.

Snyder, M. W., Adey, A., Kitzman, J. O., and Shendure, J. (2015b). Haplotype-resolved genome sequencing: experimental methods and applications. *Nature Reviews Genetics*, 16(6):344–358.

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.

Stankiewicz, P. and Lupski, J. R. (2010). Structural variation in the human genome and its role in disease. *Annual review of medicine*, 61:437–455.

Tewhey, R., Bansal, V., Torkamani, A., Topol, E. J., and Schork, N. J. (2011a). The importance of phase information for human genomics. *Nature reviews. Genetics*, 12(3):215.

Tewhey, R., Bansal, V., Torkamani, A., Topol, E. J., and Schork, N. J. (2011b). The importance of phase information for human genomics. *Nature Reviews Genetics*, 12(3):215–223.

Todd, J. A. (1933). A combinatorial problem. *Studies in Applied Mathematics*, 12(1-4):321–333.

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.

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.

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.

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.

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.

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

List of Figures

List of Tables

List of Algorithms