Department of Computer Science
Algorithmic Bioinformatics

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Solving the maximum allele co-occurrence problem for single individual haplotype phasing with Integer Linear Programming

Sarah Schweier

Bachelor Thesis

Submission:	11.03.2020
Supervisor:	Univ.-Prof. Dr. G. Klau
Second Assessor:	Dr.phil. A. Dilthey
Advisor:	S. Schrinner, M.Sc.

Abstract

Single individual diploid haplotype phasing is relevant to several applications in current research. The goal of such phasing is to identify two haplotypes that represent the sequences of the two sets of homologous chromosomes of a diploid individual. Several methods for haplotype phasing have been developed. A new approach is the Maximum Allele Co-occurrence (MAC) function, which uses sequencing read data. To phase, the function calculates two heterozygous haplotypes. These maximise, over all reads, the count of variant pairs within a read that share the alleles with one of the haplotypes at the same positions. Alberto Magi, who proposes the MAC function, solves it heuristically with his algorithm MAtCHap. In this thesis, we solve the MAC function exactly by formulating an Integer Linear Programming problem. We also formulate an ILP based on an adapted version of the MAC function that allows homozygosity. We implement the ILPs and test the running times of these implementations with synthetic and real data. We also assess the quality of the phasings of real data in contrast to MAtCHap and WhatsHap, a state of the art phasing tool. The results show, that our ILP implementations of the MAC function with and without allowed homozygosity can only phase smaller regions of a chromosome within a feasible running time. It seems, that the MAC function with allowed homozygosity produces good phasings, but the other results concerning the quality of the phasings are ambiguous.

Contents

1 Introduction

The process of identifying two haplotypes of a diploid individual is called single individual diploid haplotype phasing. Each haplotype represents the sequences of the chromosomes, a diploid organism has inherited from one parent. The phased haplotypes are relevant for various applications [4], such as understanding the relationships between DNA sequence and diseases [13]. Several approaches to haplotype phasing exist. For example, WhatsHap [11] is a tool that phases haplotypes with sequencing read data as input. WhatsHap uses a dynamic programming algorithm, which solves the weighted Minimum Error Correction problem. The idea of Minimum Error Correction (MEC) is to partition the reads into two sets and calculate two haplotypes that correspond to one set each. The sets and haplotypes are calculated such that the total count of corrections within the reads, needed to make the reads fit the assigned haplotype, is minimized. In the weighted version, a function assigns a weight to every position in each read. The weight represents a score of confidence in the correctness of the allele at this position. Here, the added sum of all weights of all corrected positions is minimized [12]. There are also other algorithms, which are based on MEC. Examples are an exact branch-and-bound algorithm and a genetic algorithm, which approximates MEC [14].

In this thesis, we explore another approach to diploid individual haplotype phasing using sequencing read data. Alberto Magi considers, instead of correcting mistakes like in MEC, the idea that a pair of alleles occurring together within a read implies that it should also occur together in one haplotype. He formalizes this with the Maximum Allele Co-Occurrence (MAC) function [10], which calculates the total amount of allele co-occurrences for two heterozygous haplotypes. To find the haplotypes that maximise the MAC function, Magi proposes an algorithm, called MAtCHap. Magi does not show, that MAtCHap solves the MAC function exactly. We, therefore, assume that it is a heuristic algorithm.

To see how well the MAC function can phase haplotypes we constructed and implemented an Integer Linear Programming problem to maximise the MAC function optimally. Further, we explored an adapted version of the MAC function that allows homozygosity between the two haplotypes. We tested our implementations with synthetic data to compare their running times to the times of WhatsHap and MAtCHap. Real data was used to assess the running times and the quality of the phasings, also in comparison to WhatsHap and MAtCHap. To measure the quality, we compare the phasing results to trusted phasing data. Two criteria, the Hamming rate, and the switch error rate are used for evaluation [11].

We observe, that it is only feasible to phase smaller regions of a chromosome with our ILP implementations of the MAC function with and without allowed homozygosity. The results for the phasing qualities were mostly ambiguous, but the implementation of the MAC function with allowed homozygosity seems to yield good results. Some expectations we had were challenged by our results.

2 Preliminaries

In this section, we describe the concepts needed for this thesis. Firstly, we define haplotype phasing and related biological terms. As these are sometimes defined differently, the scope of these definitions is restricted to this work. Since we consider diploid haplotype phasing, we only define the concepts for diploid individuals. Secondly, we define Integer Linear Programming problems, which we use to phase the haplotypes.

2.1 Biological Background

Term	**Definition**
DNA	The genetic information of organisms is stored in long molecules as a sequence of bases: Adenine, Cytosine, Thymine, Guanine. These molecules are called DNA and are contained in most cells. The sequence is usually abstracted as a string of A, C, T, and G.
Chromosome	A chromosome is a single DNA molecule, which codes specific characteristics of an organism.
Locus	A locus is a location on a chromosome.
Gene	A gene is a unit that codes a certain characteristic. It is located at a locus on a chromosome.
Allele	An allele is a specific sequence of any length in an organism's DNA. Often, allele refers to either a single base that might differ between organisms or the sequence an organism has for a gene.
Genome	The set of all genes of a species is called a genome.
Diploid	Organisms that have two sets of so-called homologous chromosomes are called diploid. Two homologous chromosomes contain the same genes but not always the same alleles for these genes since the chromosomes are inherited from different parents.
Genotype	The pairs of alleles for at least one gene or any length of a sequence of an organism are called a genotype.
Haplotype	A haplotype is the alleles for at least one gene or any length of a sequence of an organism that are inherited from a single parent. Therefore, it is a specific sequence of only one of the homologous sets of chromosomes.
Variant	A variant is a position on a chromosome, where the two haplotypes are expected to differ.

Reference Genome	A reference genome is a synthetic sequence that represents an average haplotype of a species.

2.1.1 Haplotype Phasing

Single individual diploid haplotype phasing is the process of determining the two haplotypes of an organism. In this thesis, we consider haplotype phasing that is done with sequencing read data. This data consists of reads, sequences of DNA-fragments, which have been determined by sequencing the organism's DNA. They have been aligned to a reference genome.

From a computational perspective, haplotype phasing can be defined as follows. Given a set of aligned reads, find the two haplotypes that represent those reads best and are therefore close to the original haplotypes, from which the reads were generated. Models like the MAC function [10] and MEC [12] are used to define what "representing the reads best" means, since this has not been established, yet. Under perfect conditions, haplotype one represent one half of the reads and haplotype two the other half of the reads. Furthermore, there would be no differences between the haplotypes and the reads they represent.

Concerning the input reads, it is feasible to only consider positions, in which the haplotypes are expected to differ. Therefore, the input reads can be shortened to only contain the covered variants. These positions are found via a process called variant calling. The variants are saved in a matrix with a row for every read, and a column for every variant. To simplify, the two possible alleles for a variant are represented as "0" for the reference allele and "1" for the alternative allele. When a variant is not covered by a read, this is symbolized by a "$-$" [10].

2.2 Integer Linear Programming

The following definition is based on the book "Introduction to Mathematical Optimization" by Matteo Fischetti [6], with some simplification to match the scope of this thesis.

A Linear Programming problem (LP) consists of a linear function, called the objective function, and a finite set of linear constraints. The objective function is maximized or minimized with respect to the constraints. Conventionally, LPs are formalized as minimization problems. But since the problem in this thesis is formulated as a maximization problem, we define an LP as:

$$\max\{c^T x : \; Ax \geq b, \; x \geq 0\},$$

where $x \in \mathbb{R}^n$ describes a possible solution, $c \in \mathbb{R}^n$ is a constant vector and $c^T x$ the objective function, which is maximized. The matrix $A \in \mathbb{R}^{m \times n}$ and the vector $b \in \mathbb{R}^m$ are constant. $Ax \leq b$ describes all m linear constraints. $x \in \mathbb{R}^n$ is called a feasible solution, if $x \geq 0$ and x satisfies the constraint $Ax \geq b$. A solution $x^* \in X$ is optimal, if it maximizes the objective function.

We can convert minimization problems to this form by multiplying the objective function by -1. Constraints of the form $Ax = b$ and $Ax \geq b$ can also be transformed to match the definition. It is often easier to write down an LP in a more readable way that does not completely conform to the definition. For example, we write the constraints as functions instead of giving the matrix and vectors. It has to be possible to transform the LP to match the definition.

Integer Linear Programming problems (ILPs) are similar to Linear Programming problems, but all variables are restricted to be integers: $x \in \mathbb{Z}^n$. There are no algorithms to solve ILPs in polynomial time and the decision problem for an ILP is NP-complete [7].

3 The Maximum Allele Co-occurrence model

In this section, we discuss the Maximum Allele Co-occurrence function, a new approach to haplotype phasing, which Alberto Magi proposed [10].

Reads that cover more than one variant give information about which alleles occur together in each haplotype unless there have been sequencing errors. Magi uses this concept for the MAC function. He states, that if a pair of variants has the same pair of alleles on many reads, it is likely that the variants have those alleles in one haplotype. We, therefore, try to find two haplotypes, that maximise the total number of times that two variants on a read share the alleles with one haplotype at the same position. The total count of co-occurrences is calculated by the allele co-occurrence objective function. By maximizing this function, we find the best possible haplotypes. The input for the objective function is a set of reads, which have been aligned to a reference genome and contain only variants, as discussed in Chapter 2.1.1. These reads are stored in a matrix M, which has a column for every variant and a row for every read. If a read i covers a variant j, $M[i, j]$ contains the allele at that position. $M[i, j]$ is set to 0 for the reference allele and to 1 for the alternative allele. For all variants j in a read i, which are not covered, we set $M[i, j]$ to "$-$".

Magi defines the objective function as follows:

$$MAC(M, H) = \sum_{i=1}^{n} \sum_{j \in \{M[i,j] \neq -\}} \sum_{k \in \{M[i,k] \neq -\}} \delta\left(\left(M[i, j], M[i, k]\right), \left(h_1[j], h_1[k]\right)\right) +$$
$$\delta\left(\left(M[i, j], M[i, k]\right), \left(h_2[j], h_2[k]\right)\right)$$

$$\text{with } \delta\left(\left(x_1, y_1\right), \left(x_2, y_2\right)\right) = \begin{cases} 1, & \text{if } x_1 = x_2 \text{ and } y_1 = y_2 \\ 0, & \text{otherwise} \end{cases}$$

The input matrix M has the dimension $n \times m$, where n is the number of reads, and m is the number of variants. The haplotypes are $H = (h_1, h_2)$. Magi assumes heterozygosity, that is $h_1[j] \neq h_2[j]$ for $j \in \{1, \ldots, m\}$. For each read i, the objective function compares the alleles of all pairs of variants j and k, which are covered by the read, to the alleles of both of the haplotypes at the same positions. If for either haplotype both alleles are equal, the number of the total allele co-occurrence is increased by one. In the following, "MAC" will be used to refer to this function.

To find the two haplotypes that maximise the MAC function, Magi proposes an algorithm, called MAtCHap. The algorithm calculates the haplotypes by iterating over the variants. For each variant, both possible assignments of alleles to the haplotypes are considered, and the assignment that maximises a variant-specific MAC function is chosen. This is repeated until the haplotypes are not changed in one iteration [10].

3.1 Allowing Homozygosity

Magi assumes that the haplotypes differ in each position. This is called a heterozygous assumption. If we assume that there are variants in the input data at which the haplotypes have the same alleles, we want to allow homozygosity. The allele co-occurrence objective function with allowed homozygosity does not give the desired output.

The idea of the MAC function is to consider all pairs of variants j and k in each read i. We want to maximize the count of pairs that match the alleles of one haplotype at the same positions. The previous formulation rewards a pair twice if its alleles co-occur with both haplotypes. If there is a haplotype h^* that has more allele co-occurrences with the reads than any other possible haplotype, the computed haplotypes are both h^*.

To allow homozygous positions, but still get a sensible output, we only count one co-occurrence for each two covered variants within a read. Instead of adding the outputs of the δ-functions, we combine them with a logical or. This leads us to the following definition:

$$MAC_{hom}(M,H) = \sum_{i=1}^{n} \sum_{j\in\{M[i,j]\neq -\}} \sum_{k\in\{M[i,k]\neq -\}} \delta\left((M[i,j],M[i,k]),(h_1[j],h_1[k])\right) \vee$$
$$\delta\left((M[i,j],M[i,k]),(h_2[j],h_2[k])\right)$$

In the following, "MAC_{hom}" will be used to refer to this function.

4 The MAC function as an Integer Linear Programming problem

Magi used MAC for an algorithm, called MAtCHap [10], which we assume to be a heuristic.

In this thesis, we formulate MAC as an ILP to explore a way to solve it exactly. The ILP model follows MAC closely. The variables h_j for $j \in \{1,\ldots,m\}$ represent one haplotype. Since we assume heterozygosity, we do not need variables for the second haplotype. It can be calculated by taking $1-h_j$ for the j-th allele. The set V_i contains all variants, which are covered by the read i. For each read i and each variant $j \in V_i$, the variables $e_{1,i,j}$ and $e_{2,i,j}$ compare the allele at position j on read i to the j-th allele of haplotypes one and two, respectively. If the alleles are the same, $e_{1,i,j}$ is set to 1, otherwise to 0. The same is true for

$e_{2,i,j}$. The output of each δ-function in the MAC objective function is modelled as its own variable. For each read i and each pair of variants $j,k \in V_i$, the variables $\delta_{1,i,j,k}, \delta_{2,i,j,k}$ describe whether the alleles at j and k are equal to the alleles at j and k in haplotypes one and two, respectively. The constraints realize the described properties. $\delta_{1,i,j,k}$ is modelled using $e_{1,i,j}$ and $e_{1,i,k}$. If $e_{1,i,j}$ and $e_{1,i,k}$ are equal to 1, the alleles at positions j and k agree between read i and haplotype one and therefore $\delta_{1,i,j,k}$ is set to 1. This also holds for $\delta_{2,i,j,k}$, $e_{2,i,j}$ and $e_{2,i,k}$. The objective function sums up all δ variables. The ILP is defined as follows:

$$\begin{array}{lll}
\text{max} & \displaystyle\sum_{i=1}^{n} \sum_{j \in V_i} \sum_{k \in V_i} \delta_{1,i,j,k} + \delta_{2,i,j,k} & \\
\text{s.t.} & \delta_{1,i,j,k} = e_{1,i,j} \wedge e_{1,i,k} & \forall i \in \{1,\dots,n\};\ \forall j,k \in V_i \\
& \delta_{2,i,j,k} = e_{2,i,j} \wedge e_{2,i,k} & \forall i \in \{1,\dots,n\};\ \forall j,k \in V_i \\
& e_{1,i,j} = h_j == M[i,j] & \forall i \in \{1,\dots,n\};\ \forall j \in V_i \\
& e_{2,i,j} = 1 - h_j == M[i,j] & \forall i \in \{1,\dots,n\};\ \forall j \in V_i \\
& & \\
& \delta_{1,i,j,k}, \delta_{2,i,j,k} \in \{0,1\} & \forall i \in \{1,\dots,n\};\ \forall j,k \in V_i \\
& e_{1,i,j}, e_{2,i,j} \in \{0,1\} & \forall i \in \{1,\dots,n\};\ \forall j \in V_i \\
& h_j \in \{0,1\} & \forall j \in \{1,\dots,m\}
\end{array}$$

$$\text{with } V_i = \{j \mid j \in \{1,\dots,m\},\ M[i,j] \neq -\}\ \forall i \in \{1,\dots,n\}$$

The operations $\wedge$ and $==$ are not defined for ILPs, but they can easily be transformed to comply with the definition from Chapter 2.2 as follows. Let $z, a, b \in \{0,1\}$.

$$z = a \wedge b \iff \begin{array}{l} z \leq a \\ z \leq b \\ z \geq a + b - 1 \end{array}$$

$$z = a == b \iff \begin{array}{l} z \leq a - b + 1 \\ z \leq b - a + 1 \end{array}$$

The $==$ operation is missing the constraints that set lower bounds. This is sufficient since we maximize and therefore try to set as many $e_{1,i,j}$ and $e_{2,i,j}$ to 1 as possible. For the $\vee$ operation the last constraint can be left out for the same reasons.

4.1 Allowing Homozygosity

The ILP formulation for MAC_{hom}, which allows homozygosity, is similar to the previous definition. We need to add variables for the second haplotype, since it cannot be derived from the first haplotype anymore. Here, $h_{1,j}$ stands for the allele in haplotype one at variant j and $h_{2,j}$ does the same for haplotype two. Since the variables $\delta_{1,i,j,k}$, $\delta_{2,i,j,k}$ have to be connected by a logical or, we need additional variables $\delta_{i,j,k}$ for $i \in \{1, \dots, n\}$; for $j, k \in V_i$. If the alleles at the variants j and k in read i occur together in a least one of the haplotypes, $\delta_{i,j,k}$ is set to 1, otherwise to 0. The ILP is defined as follows:

$$
\begin{array}{lll}
\text{max} & \displaystyle\sum_{i=1}^{n} \sum_{j \in V_i} \sum_{k \in V_i} \delta_{i,j,k} & \\
\text{s.t.} & \delta_{i,j,k} = \delta_{1,i,j,k} \vee \delta_{2,i,j,k} & \forall i \in \{1, \dots, n\};\ \forall j, k \in V_i \\
 & \delta_{1,i,j,k} = e_{1,i,j} \wedge e_{1,i,k} & \forall i \in \{1, \dots, n\};\ \forall j, k \in V_i \\
 & \delta_{2,i,j,k} = e_{2,i,j} \wedge e_{2,i,k} & \forall i \in \{1, \dots, n\};\ \forall j, k \in V_i \\
 & e_{1,i,j} = h_{1,j} == M[i,j] & \forall i \in \{1, \dots, n\};\ \forall j \in V_i \\
 & e_{2,i,j} = h_{2,j} == M[i,j] & \forall i \in \{1, \dots, n\};\ \forall j \in V_i \\
 & & \\
 & \delta_{i,j,k} \in \{0,1\} & \forall i \in \{1, \dots, n\};\ \forall j, k \in V_i \\
 & \delta_{1,i,j,k}, \delta_{2,i,j,k} \in \{0,1\} & \forall i \in \{1, \dots, n\};\ \forall j, k \in V_i \\
 & e_{1,i,j}, e_{2,i,j} \in \{0,1\} & \forall i \in \{1, \dots, n\};\ \forall j \in V_i \\
 & h_{1,j}, h_{2,j} \in \{0,1\} & \forall j \in \{1, \dots, m\}
\end{array}
$$

$$
\text{with } V_i = \{j \mid j \in \{1, \dots, m\},\ M[i,j] \neq -\} \ \forall i \in \{1, \dots, n\}
$$

The operation $\vee$ is also not defined for ILPs, but it can easily be transformed to comply with the definition from chapter 2.2 as follows. Let $z, a, b \in \{0, 1\}$.

$$
z = a \vee b \iff \begin{array}{l} z \geq a \\ z \geq b \\ z \leq a + b \end{array}
$$

5 Implementation

We used Python 3.6.7 for all code in this thesis. The code can be found in this work's GitLab repository linked in Appendix A. We implemented the ILPs using PuLP 1.6.0[1], a python package developed for optimization. We call CPLEX[2], a commercial solver, with PuLP to solve the ILPs. Our implementation takes a list of dictionaries as input. Each dictionary represents a read and contains all covered variants as keys, and the alleles at these variants as values. The output is the two computed haplotypes and the calculated MAC Score, the total count of allele co-occurrences.

[1] https://pythonhosted.org/PuLP/index.html#

[2] www.cplex.com

To be able to process BAM and VCF files as input we call our code from within the developer version of WhatsHap[3] [11]. Whatshap is discussed in Chapter 1. WhatsHap's output is a VCF file. We added two command line parameters to be able to choose one of our implementations to phase instead of the native WhatsHap implementation. We described the detailed integration into WhatsHap in a ReadMe in the Git Lab Repository. The changes in the WhatsHap code were made by Sven Schrinner. To phase with the ILP implementation that allows homozygosity, the flag "–distrust-genotypes" needs to be added to the WhatsHap call. We call all phasings with "–ignore-read-groups" to avoid errors due to read group metadata not matching between BAM and VCF input files.

To phase with MAtCHap, we need to create fragment files from the BAM and VCF files, since MAtCHap [10] takes a VCF and a fragment file as input. To create the fragment file, we use the extractHAIRS command of HapCut2 [5], another tool for haplotype assembly. MAtCHap's output is a VCF file.

To compare the output VCF files to our trusted haplotypes and therefore assess the phasings, we used WhatsHap's compare command. To do so, we needed to add the first line that specifies the file format to the output VCF of MAtCHap in order to make it readable for WhatsHap.

The testing was done using a Snakemake workflow [8]. Snakemake is a tool for implementing workflows that uses a Python-based language. To specify a workflow, rules are defined, which depend on each other. A rule consists of one or more input files, one or more output files, and a command. Wanted output files are specified and rules call each other to create them.

6 Results

In this section, we show how our implementations perform and compare them to WhatsHap and MAtCHap. We used synthetic data to test how our implementation's running time scales. Real data was used to compare the running times of our implementations, WhatsHap [11] and MAtCHap [10] and to assess the quality of the phasings.

6.1 Synthetic Data

6.1.1 Generation

We generated synthetic data by sampling reads from random haplotypes. Our implementation takes the number of variants, the coverage, the minimum and maximum length of the reads, the rate of errors in the reads and the rate of homozygosity between the haplotypes as input. First, the implementation generates one haplotype by filling positions randomly with zeros and ones. The other haplotype is generated by inverting the first haplotype, with the given probability of homozygosity. After this, reads are generated from both haplotypes with random starting positions and random lengths within the given bounds. Errors are placed with the given probability. This is repeated until the

[3] https://whatshap.readthedocs.io/en/latest/develop.html

average coverage is surpassed. Each read is saved as a dictionary, where the keys are the covered positions and the values are the alleles at these positions.

To test the running time of the implementations of MAC and MAC_{hom}, we generated 300 data sets for each of the implementations. The specifications of these data sets included all combinations of coverages $5\times$, $10\times$, and $15\times$, error rates of 0%, 2%, 5%, and 10%, and numbers of variants from 20 to 500 with a step size of 20. The minimum and maximum length of reads were set to 6 and 14, respectively. These are realistic counts of variants for a read, since the human genome, which consists of 3.2 billion base pairs (bp) [3], contains 4 to 5 million variants [2], that is more than one variant every 1000 bp, and long reads can cover 10000 bp [1]. The homozygosity rate was set to 0. We generated five data sets for each combination of specifications.

To test the differences between the total number of allele co-occurrences calculated by the implementations of MAC and MAC_{hom} we generated one dataset for each heterozygosity rate from 0% to 20% with a step size of 1. The number of variants was set to 100, the coverage to $15\times$ and the error rate to 0%. The minimum and maximum length of reads were set to 6 and 14, respectively.

6.1.2 Running time

The tests were run on the High-Performance Cluster of the Heinrich Heine Universität Düsseldorf with 16 CPUs and 16GB of memory. We solved the random data sets mentioned in section 6.1.1 with the ILP implementations of MAC and MAC_{hom}. It should be noted that new data was generated for each implementation with the same specifications. We computed the average time to solve the five data sets with the same specification for each implementation. The allowed maximum for the average running time was set to 15 minutes. If an average running time exceeded 15 minutes for a count of variants, this running time was discarded. The running times for a higher amount of variants and the same coverage and error rate were not calculated, as they are expected to take longer.

The measured running times for the implementation of MAC can be found in Figure 1. The plots top left, top right, bottom left and bottom right represent the results for error rates 0%, 2%, 5%, and 10%, respectively. Each plot shows the measured running times for coverage $5\times$, $10\times$, $15\times$. The running times were calculated for 20 to 500 variants with step size 20, but only if the average running time did not exceed 15 minutes. That means, that when a curve ends before 500 variants, the average running time for the next count of variants exceeded 900 seconds and the following running times were not calculated. When a curve ends at 500 variants, there is no information about the running times for higher counts of variants.

The plots show that the running times mostly increase with the count of variants, with some exceptions. We did not expect the running time to decrease as much as it does for coverage $5\times$ and error rate 10%, even though it rises again after 200 variants. This might be related to the high error rate. Starting at 80 variants, a higher coverage always yields a higher running time. We expected the increase with higher coverage and more variants since these signify a larger input. A higher error rate also increases the running times. We expected this because we assumed phasing with a higher error rate to be more complex.

The running times measured for the implementation of MAC_{hom} are shown in Figure 2. They display similar properties to the running times for the implementation of MAC, while always being higher for coverage $10\times$ and $15\times$ and mostly for coverage $5\times$. This was expected since MAC_{hom} allows more possible haplotypes and therefore has a larger search space.

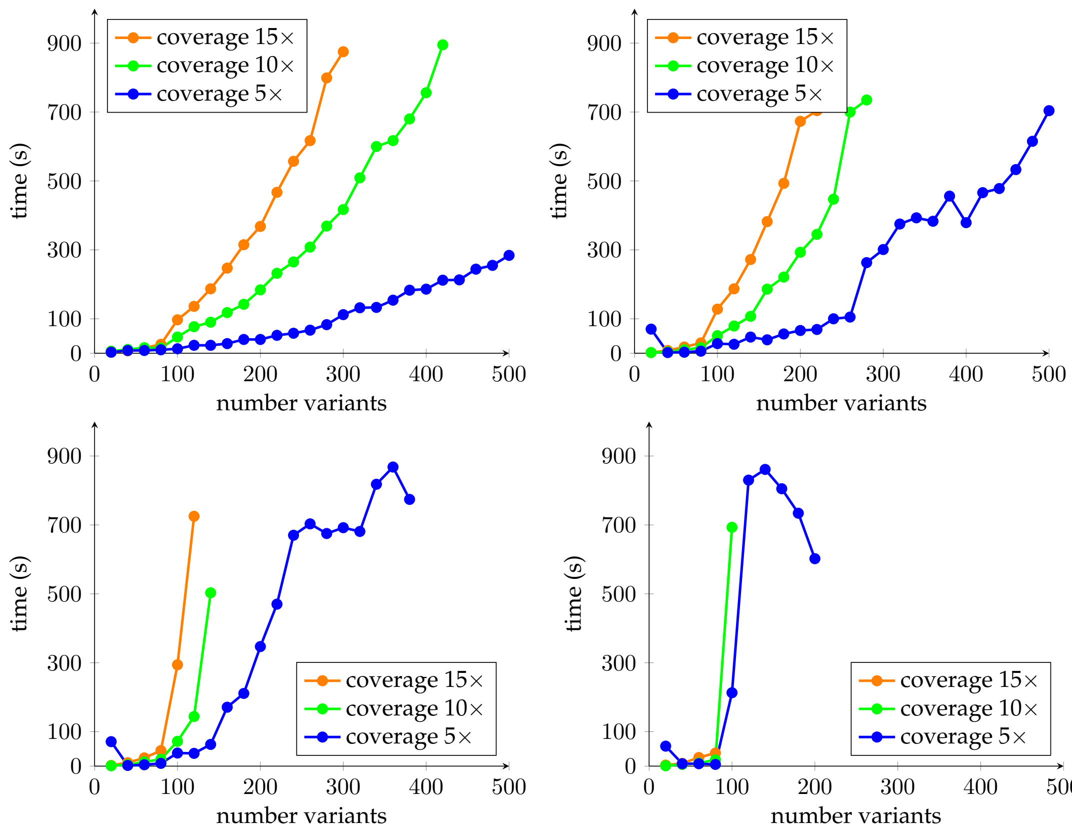


Figure 1: These plots show the average running time of the implementation of MAC measured for five random data sets for 20 to 500 variants with stepsize 20, coverage $5\times$, $10\times$ and $15\times$ and error rate 0% (top left), 2% (top right), 5% (bottom left), 10% (bottom right). When an average running time exceeded 900 seconds, it was discarded and the running times for more variants but the same coverage and error rate were not calculated.

6.1.3 Score

The tests for the score differences between the implementations of MAC and MAC_{hom} were run on the High-Performance Cluster of the Heinrich Heine Universität Düsseldorf with 16 CPUs and 10GB of memory. We solved the random data sets mentioned in the last paragraph of Chapter 6.1.1 with the implementations of MAC and MAC_{hom}. Here, we solved each data set with both implementations. To quantify the score improvement caused by allowing homozygosity, we calculated the percentage of difference between the scores of MAC and MAC_{hom}. More precisely, we calculated the absolute difference between the scores of MAC and MAC_{hom} and then divided by the score of MAC.

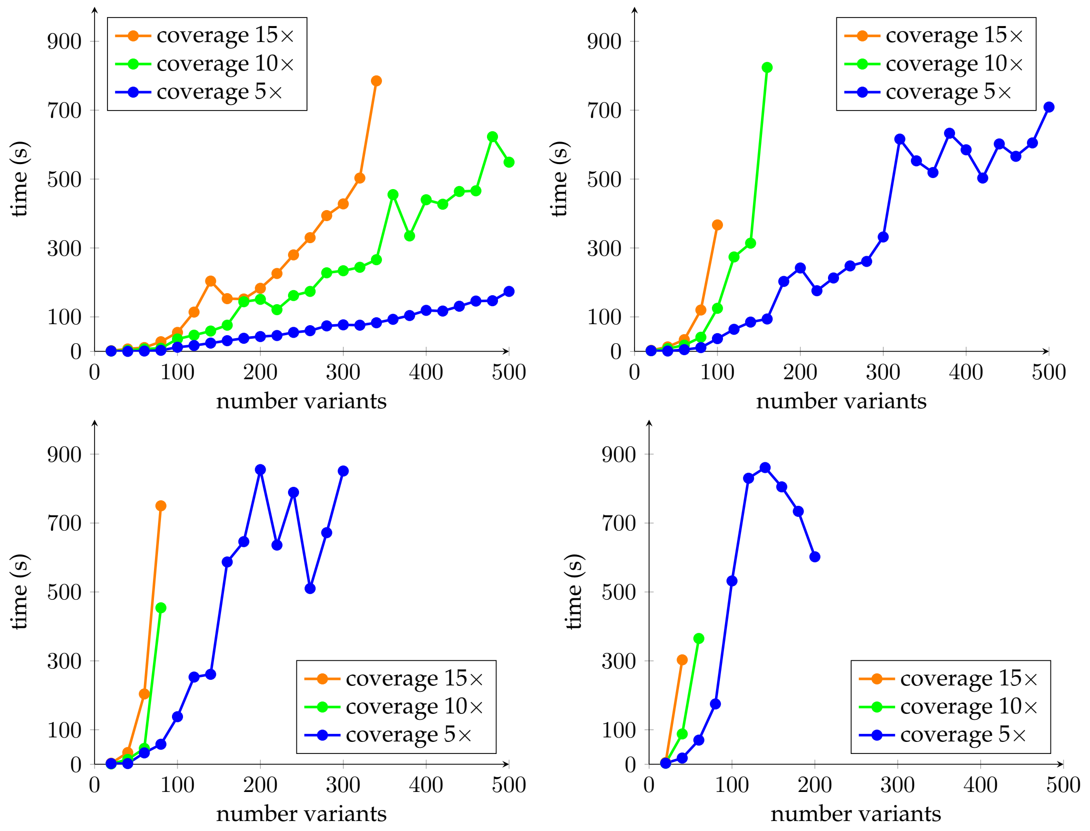


Figure 2: These plots show the average running time of the implementation of MAC_{hom} measured for five random data sets for 20 to 500 variants with stepsize 20, coverage 5×, 10× and 15× and error rate 0% (top left), 2% (top right), 5% (bottom left), 10% (bottom right). When an average running time exceeded 900 seconds, it was discarded and the running times for more variants but the same coverage and error rate were not calculated.

The measured rates of score differences are shown in Figure 3. As expected, the rate rises when the rate of homozygosity increases. With more homozygous positions in the haplotypes, we expect the implementation of MAC_{hom} to compute haplotypes that have more allele co-occurrences with the reads than the haplotypes computed by the implementation of MAC.

6.2 Real Data

6.2.1 Data

We used the same data, which was used by Magi for the evaluation of his proposed tool MAtCHap. [10]. We downloaded the BAM file and BAM.BAI file from `ftp://ftp-trace.ncbi.nih.gov/giab/ftp/data/NA12878/NA12878_PacBio_MtSinai/`. Magi states that the BAM file contains PacBio reads from the genome NA12878 from the CEPH Utah Reference collection [15] created by the Genome

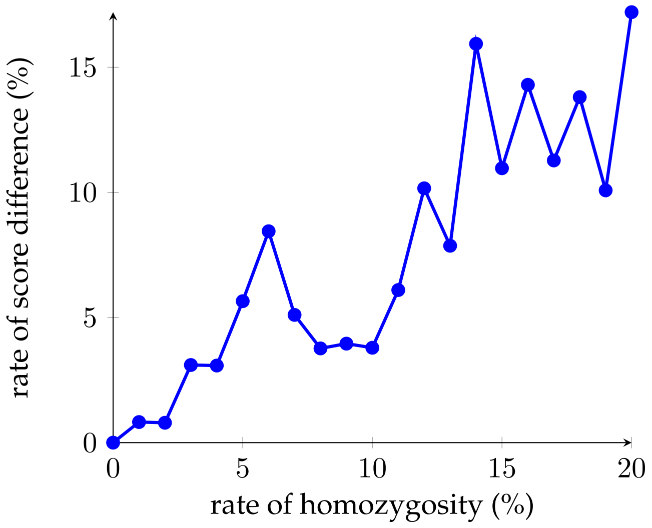


Figure 3: This plot shows the rates of differences between the scores calculated for MAC and MAC_{hom} for random data sets with 100 variants, error rate 0% and rates of heterozygosity from 0% to 20% with stepsize 1.

in a Bottle Consortium[4]. According to him, they have a total sequencing coverage of 45× and were mapped to the human reference genome (hg19) with BLASR (v1.3.2). Since we could not find the given link on the GitHub page of the Genome in a Bottle Consortium, we were not able to confirm this. We downloaded the VCF file from `ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/NA12878_HG001/latest/GRCh37/`. Magi states, that the VCF file contains high confidence variants identified by the Genome in a Bottle Consortium [10]. We also could not confirm this.

All terminal commands mentioned here can be found in the snakefile in the Git Lab Repository linked in Appendix A. Since our implementation cannot phase a whole chromosome within a reasonable time frame, we used smaller regions. To cut the VCF file, we used tabix[5], a command line tool from samtools [9]. To cut the BAM file, we also used samtools. The regions that we cut from chromosomes one and two are shown in Table 1. The labelling of the chromosomes differs between the VCF files and the BAM files. We adjusted the labelling of the VCF files, which is the number of the chromosome, to the labelling of the BAM files, which is 'chr' followed by the number of the chromosome. Additionally, in the VCF files, all but the GT column from the format field and the corresponding columns from the HG001 field were removed, as they are not needed, and WhatsHap has problems interpreting to contents of the PS column. We used samtools to index the BAM files. We measured the coverage of the BAM files with samtools depth and awk[6] and downsampled the BAM files to coverage 10× and 15× using samtools view -s with seed 0, which makes the downsampling repeatable. These BAM files were also indexed with samtools index. Since we do not know which seed Magi used, his downsampled BAM files might be different.

[4]`https://github.com/genome-in-a-bottle`
[5]`https://www.htslib.org/doc/tabix.html`
[6]`https://pubs.opengroup.org/onlinepubs/9699919799/utilities/awk.html`

Region	Count Variants Chr. 1		Count Variants Chr. 2	
	coverage 10×	coverage 15×	coverage 10×	coverage 15×
60000000-60075000	52	52	82	82
60000000-60100000	58	59	106	106
60000000-60125000	74	74	117	118
60000000-60150000	90	90	119	120
60000000-60175000	114	114	119	120
60000000-60200000	130	130	123	124

Table 1: Count of variants phased for each region and coverage.

WhatsHap's preprocessing, which is used by WhatsHap itself and the ILP implementations of MAC and MAC_{hom}, downsamples BAM files to a given maximum coverage for each position. The largest possible coverage is 25×, which we chose to avoid the BAM files to be downsampled further. Nevertheless, for all regions of chromosome two at coverage 15×, one read was removed by WhatsHap, since samtools only downsamples to an average coverage. This means that MAtCHap phases these regions with one more read. The variants that are covered by at least one read that also covers another variant can be used for phasing. The count of these variants for each region can be seen in Table 1. This disregards the additional read, that MAtCHap used for phasing, as the information was taken from WhatsHap's output.

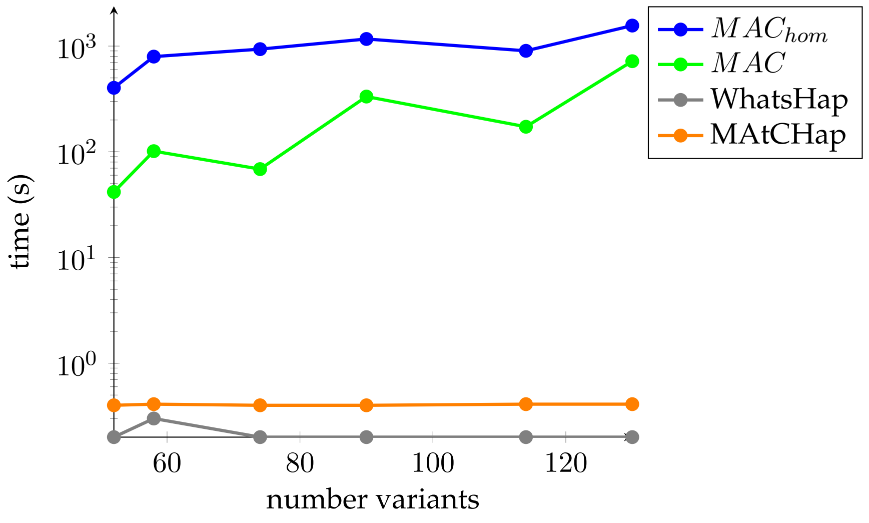
6.2.2 Running time

The phasing with MAtCHap did not work on the High-Performance Cluster of the Heinrich Heine Universität Düsseldorf. It was possible to start MAtCHap, but even for small instances, it did not terminate for a yet unknown reason. The phasing was instead done on a personal computer with four CPUs and 8GB of memory. The other phasings were run on the High-Performance Cluster of the Heinrich Heine Universität Düsseldorf with 16 CPUs and 10GB of memory.

The running time of MAtCHap was determined with the terminal command "time", which measures the total elapsed time. We assume that the phasing time of MAtCHap is not significantly shorter than the total elapsed time. We used the actual phasing time as running time for WhatsHap and the ILP implementations of MAC and MAC_{hom}.

The measured running times for chromosome one are shown in Figure 4. The top plot shows the times measured for all four implementations for the different numbers of variants with coverage 10×. The bottom plot shows the same for coverage 15×. MAtCHap has a running time of around 0.4 seconds for all regions and both coverages. This is the shortest time for coverage 15× and the second to shortest for coverage 10×. That the running time does not increase much with higher coverage, agrees with Magi's testing, where he shows that MAtCHap's running time increases slowly with higher coverage [10]. WhatsHap has the shortest running times for coverage 10×. For coverage 15× it has the second to shortest running times and they start increasing at 114 variants (region 60000000-60175000). Generally, WhatsHap's running times increase with higher coverage. This was expected since its running time grows exponentially with increasing coverage [12]. The implementation of MAC is a lot slower than the mentioned tools and its running time overall increases with more variants, as already observed for synthetic data in Chapter 6.1.2. The same is true for the implementation of MAC_{hom}, which is even slower. Phasing with the implementations of MAC and MAC_{hom} takes longer with the higher coverage of 15×. This was also observed in Section 6.1.2.

The measured running times for chromosome two can be found in Figure 7 in Appendix B. These running times have similar properties to the ones for chromosome one. Here, MAtCHap has the shortest running times for both coverages.



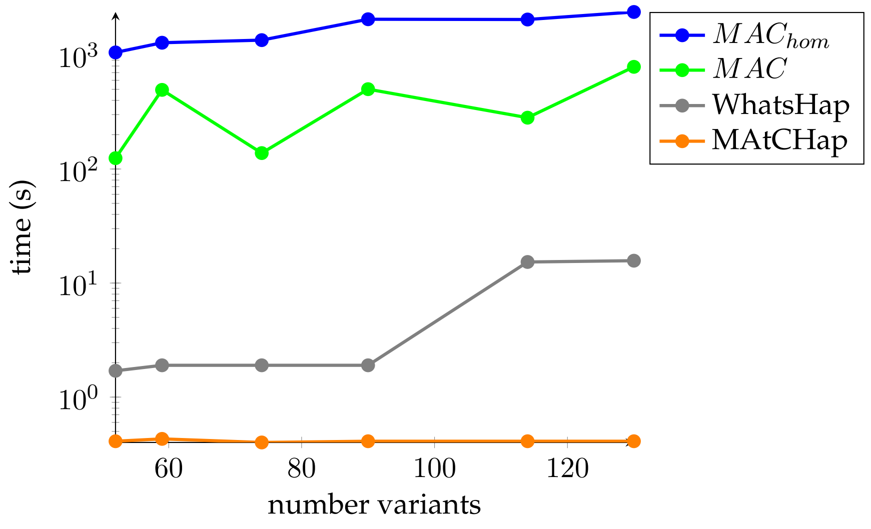


Figure 4: These plots show the running times of WhatsHap, MAtCHap and the implementations of MAC and MAC_{hom} measured for different regions on chromosome one for coverage 10× (top) and coverage 15× (bottom).

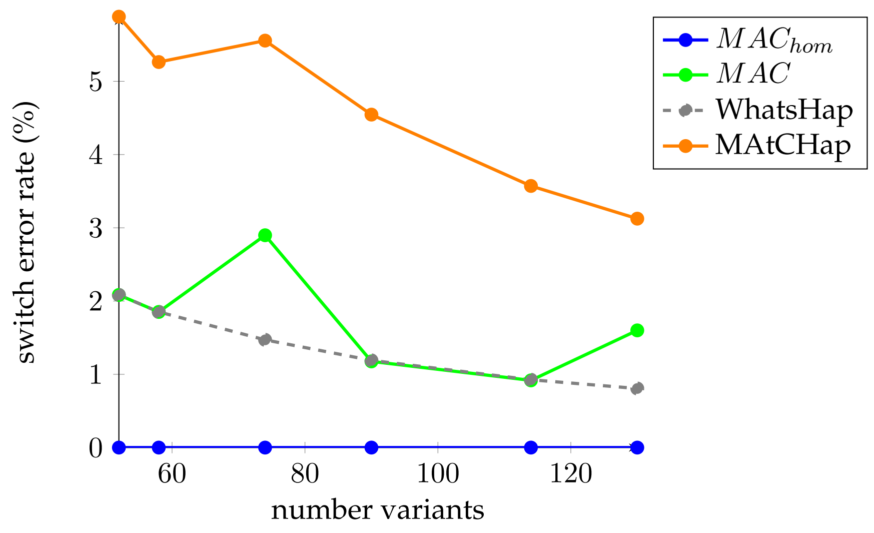
6.2.3 Quality of phasing

The quality of the phasing was measured with the command line tool compare from WhatsHap. We used the measures switch error rate and block-wise Hamming distance rate to assess the quality of the phasings. A block is a set of variants, which were phased relative to each other. That means, that for any two different blocks, the relation between the phasings of the blocks is not known. The switch error rate in a block is the number of cuts that would need to be made in this block to get the true haplotypes by rearranging the phasing regions at the cut positions, divided by the total amount of connections in the block. The switch error rate is the number of cuts needed over all blocks divided by the total amount of connections. A connection is two neighbouring variants, which have been phased relative to each other. That means that one block of length n contains $n-1$ connections and the total amount of connections is the number of phased variants minus the number of blocks. The block-wise Hamming distance rate is the sum of the Hamming distances between all blocks and the trusted phasing, divided by the total number of phased alleles. The Hamming distance is the number of phased alleles that do not agree with the trusted phasing [11].

The switch error rate for the phasings of chromosome one are shown in Figure 5. A smaller switch error rate signals a better phasing. For coverage 10×, the implementation of MAC_{hom} has switch error rates of 0% for all regions. The rates of the other three implementations overall decline slowly with an increasing number of variants. The implementation of MAC and WhatsHap have similar switch error rates, which might indicate that the MAC function and MEC are similar. MAtCHap has the highest error rates. It was expected, that the implementation of MAC phases better than MAtCHap, since the implementation of MAC solves the MAC function exactly. For coverage 15×, MAtCHap, WhatsHap and the implementation of MAC have almost the same switch error rates, which are similar to the switch error rates of WhatsHap and the implementation of MAC for coverage 10×. It was not expected that WhatsHap phases worse with higher coverage, but the difference is small. It was expected that MAtCHap phases better with higher coverage since it was designed to phase read data with high coverage. The switch error rates of the implementation of MAC_{hom} are zero for the two smallest and the largest regions but are higher for the others. They are still smaller than the error rates of all other implementations. This might be because MAC_{hom} allows homozygous variants. The increase of the switch error rates for the implementation of MAC_{hom} with higher coverage was not expected.

The switch error rate for the phasings of chromosome two are shown in Figure 8 in Appendix B. The rates have different properties than the switch error rates of chromosome one. For example, with increasing variant counts, the switch error rates increase for chromosome two and decrease for chromosome one. Further, it was not expected that the rates for the implementation of MAC are higher than the ones of MAtCHap. The rates for chromosome two do not support the interpretations made for chromosome one. The smaller switch error rates for the higher coverage were expected.

The block-wise Hamming distance rates for chromosome one are shown in Figure 6. For coverage 10×, the implementation of MAC_{hom} has rates of zero. WhatsHap has relatively small, constant rates. MAtCHap's rates increase with the number of variants.



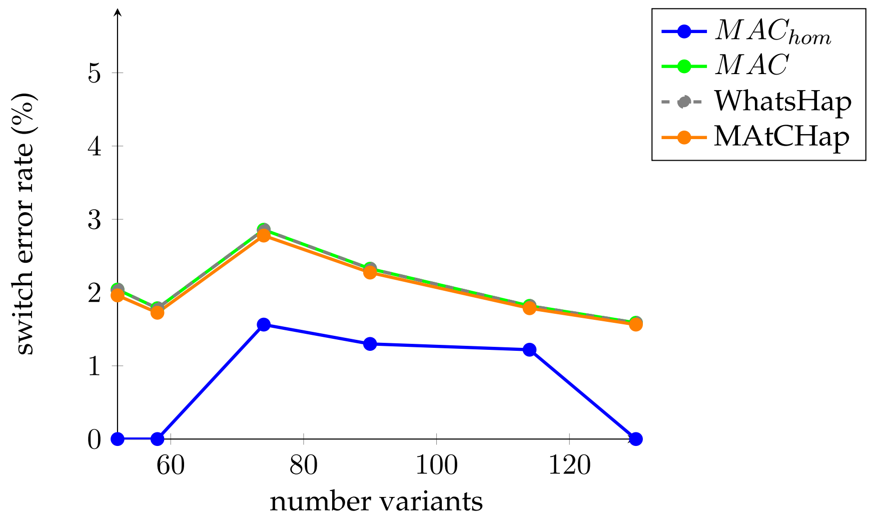
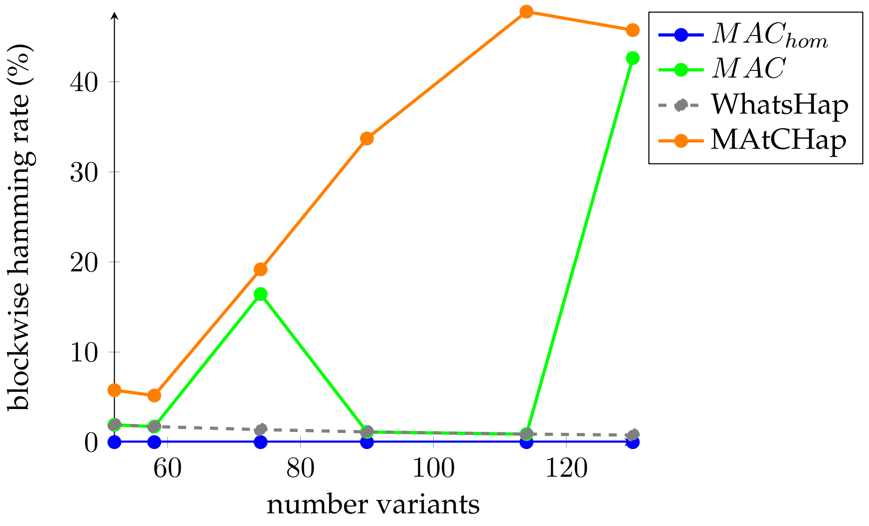


Figure 5: These plots show the switch error rates of WhatsHap, MAtCHap and the implementations of MAC and MAC_{hom} measured for phasings of different regions on chromosome one for coverage 10× (top) and coverage 15× (bottom).

The implementation of MAC has the same rates as WhatsHap except for 74 variants (region 60000000-60125000) and 130 variants (region 60000000-60200000), where the rates are similar to MAtCHap. For coverage 15×, MAtCHap's rates are very similar to its rates for coverage 10×. The rates of the other tools are close to the rates of MAtCHap, except for the implementation of MAC_{hom} at 130 variants (region 60000000-60200000), where its rate is zero. It was not expected for the rates to rise as much for the higher coverage. Instead, we expected them to be smaller for a higher coverage since more information for phasing was available. MAtCHap always has equal or worse rates than the implementation of MAC. This was expected since MAtCHap is a heuristic and the implementation of MAC solves the MAC function exactly. The implementation of MAC_{hom} has the best rates, possibly because it allows homozygous variants. The implementation of MAC and

WhatsHap share many similar rates. This might indicate a relation between the MAC function and MEC.

The block-wise Hamming distance rates for chromosome two are shown in Figure 9. The rates of MAtCHap and some for implementation of MAC are a lot smaller for chromosome two compared to chromosome one with coverage 10×. All rates except for small counts of variants are a lot smaller for chromosome two compare to chromosome one for coverage 15×. All rates for chromosome two decrease with higher coverage, as expected. The implementation of MAC_{hom} has the best rates. A relation between the MAC function and MEC is not supported here.



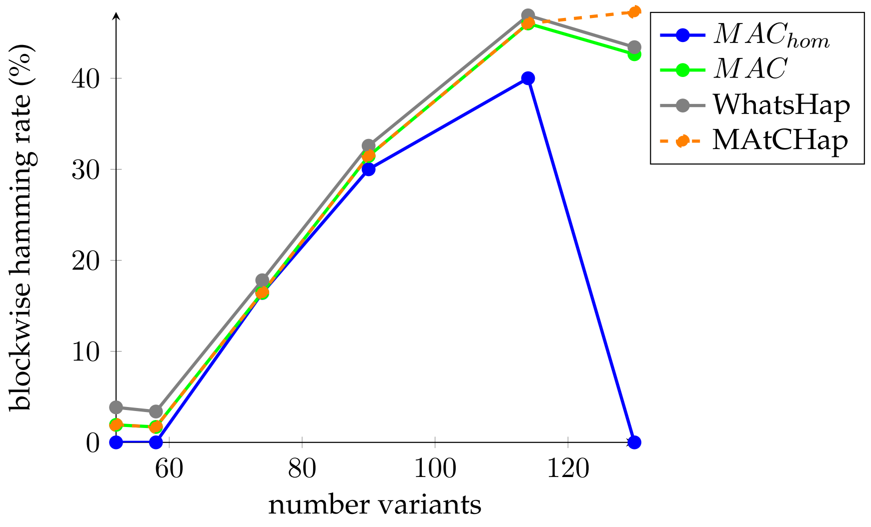


Figure 6: These plots show the block-wise Hamming distance rates of WhatsHap, MAtCHap and the implementations of MAC and MAC_{hom} measured for phasings of different regions on chromosome one for coverage 10× (top) and coverage 15× (bottom).

7 Discussion and Outlook

In this section, we discuss the results we showed in the previous chapter. We then give an outlook for further research.

The running times measured for synthetic and real data support, that an increase in the number of variants, as well as the coverage, mostly causes a higher running time for the implementations of MAC and MAC_{hom}. This was expected since a higher coverage and a larger number of variants signify a larger input. Further, an increase of the error rate also often causes a higher running time. Possibly, this is because an increased error rate makes phasing more complex. Phasing with the implementation of MAC_{hom} often takes longer than phasing with the implementation of MAC, presumably, since MAC_{hom} has a larger search space. Further, phasing with the tools MAtCHap [10] and WhatsHap [11] is around 10 to 1000 times quicker than phasing with the implementations of MAC and MAC_{hom}.

Therefore, it is only feasible to phase smaller regions of a chromosome with the implementations of MAC and MAC_{hom}. One reason for this might be the LP-Relaxation of the ILP formulations. An LP-Relaxation is an LP, that is equal to a specific ILP, but does not restrict the variables to be integers. We used CPLEX[7] to solve the ILPs. Roughly speaking, CPLEX first solves the ILP's LP-Relaxation and then finds the optimal solution of the ILP by restricting non-integer variables through branching [6]. The LP Relaxations for the ILP formulations of MAC and MAC_{hom} in Chapter 4 are easily solved by setting the alleles in both haplotypes to 0.5. This causes all $e_{1,i,j}$ and $e_{2,i,j}$ for $i \in \{1, ..., n\}$ and $j \in V_i$ to be set to 0.5. As a result all $\delta_{1,i,j,k}$ and $\delta_{2,i,j,k}$ for $i \in \{1, ..., n\}$ and $j, k \in V_i$ are set to 0.5. That means that for each pair of covered variants within a read, one co-occurrence is counted. This is the maximum value that can be achieved for any input. Since this optimal solution for the LP-Relaxation contains no integers, it might take longer to find an integer solution, since all variables need to be forced to have integer values.

As expected, the tests of the score differences show, that solving the MAC function with homozygosity allowed, yields better results for synthetic input with homozygous positions.

The measurements of the quality of the phasing for the different implementation yield ambiguous results. The phasing qualities for chromosome one suggest that the MAC function and MEC [12] could be similar, but the phasing qualities for chromosome two refute this. We expected our implementation of MAC to always yield better results than MAtCHap since our implementation solves the MAC function exactly, but this is not the case for chromosome two. Further, we expected a higher coverage to improve the phasing quality, since there is more data available, but this was not the case for the block-wise Hamming distance rates measured for chromosome one. While the block-wise Hamming distance rates increased with a larger number of variants, the switch error rates increase for chromosome one and decrease for chromosome two. Generally, there were a lot of differences between phasings of regions of the two chromosomes. It can be said, that the implementation of MAC_{hom} had the best rates in general. This might be because homozygous positions are allowed, and therefore there are more options for phasing.

[7] www.cplex.com

There are several options to further evaluate the ILP formulations for MAC and MAC_{hom}. For synthetic data, more combinations of specifications could be considered. MAtCHap does not output the total number of co-occurrences its phased haplotypes have with the input read. We could compute this number from MAtCHap's output and compare it to the number of co-occurrences calculated by the implementation of MAC, to evaluate the quality of the heuristic. For real data, other measures of quality for the phasings could be considered to compare them. Higher coverages should be considered since is designed MAtCHap to solve with them. More data sets and different chromosome regions should be used for testing to have a better base for interpretation. Further, we could try to differently formulate the MAC function as an ILP, to circumvent the problem with the LP-Relaxation.

To summarize, we implemented MAC and MAC_{hom} as ILPs. We tested these implementations with synthetic and real data and compared them to MAtCHap and WhatsHap. Our implementations have longer running times than the two tools. The phasing quality results were mostly ambiguous, but the implementation of MAC_{hom} seems to yield good results.

8 Acknowledgements

I express my gratitude to the supervisor of this thesis Prof. Dr. Gunnar Klau for giving me the opportunity to write this bachelor thesis and providing an interesting topic. I am grateful to Sven Schrinner, my advisor, for hours of support and answers to my many questions and to Dr. phil Alexander Dilthey for being this work's second assessor. I thank Jonas Weber for our fruitful discussions.

I am also grateful for the template I used, which was supplied by the research group Databases and Information Systems at the Heinrich Heine Universität Düsseldorf. I thank Justin Zoebel for his book "Writing for Computer Science", which gave great writing advice. I am grateful for the computational support and infrastructure, which was provided by the "Centre for Information and Media Technology" (ZIM) at the University of Düsseldorf (Germany).

References

[1] Kin Fai Au, Jason G Underwood, Lawrence Lee, and Wing Hung Wong. “Improving PacBio long read accuracy by short read alignment”. In: *PloS one* 7.10 (2012).

[2] A. Auton, G. Abecasis, and D. Altshuler et al. “A global reference for human genetic variation”. English (US). In: *Nature* 526.7571 (Sept. 2015), pp. 68–74.

[3] Terence A Brown. *Genomes. 2nd edition*. Wiley-Liss, 2002. Chap. 1, The Human Genome.

[4] Sharon Browning and Brian Browning. “Haplotype phasing: Existing methods and new developments”. In: *Nature reviews. Genetics* 12 (Sept. 2011), pp. 703–14.

[5] Peter Edge, Vineet Bafna, and Vikas Bansal. “HapCUT2: robust and accurate haplotype assembly for diverse sequencing technologies”. In: *Genome Research* (2016). eprint: http://genome.cshlp.org/content/early/2016/12/09/gr.213462.116.full.pdf+html.

[6] Matteo Fischetti. *Introduction to Mathematical Optimization*. Independently published, 2019.

[7] Michael R. Garey and David S. Johnson. *Computers and Intractability*. Freeman, 1979.

[8] Johannes Köster and Sven Rahmann. “Snakemake—a scalable bioinformatics workflow engine”. In: *Bioinformatics* 28.19 (Aug. 2012), pp. 2520–2522. eprint: https://academic.oup.com/bioinformatics/article-pdf/28/19/2520/819790/bts480.pdf.

[9] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, and 1000 Genome Project Data Processing Subgroup. “The Sequence Alignment/Map format and SAMtools”. In: *Bioinformatics* 25.16 (June 2009), pp. 2078–2079. eprint: https://academic.oup.com/bioinformatics/article-pdf/25/16/2078/531810/btp352.pdf.

[10] Alberto Magi. “MAtCHap: an ultra fast algorithm for solving the single individual haplotype assembly problem”. In: *bioRxiv* (2019). eprint: https://www.biorxiv.org/content/early/2019/12/02/860262.full.pdf.

[11] Marcel Martin, Murray Patterson, Shilpa Garg, Sarah O Fischer, Nadia Pisanti, Gunnar W Klau, Alexander Schöenhuth, and Tobias Marschall. “WhatsHap: fast and accurate read-based phasing”. In: *bioRxiv* (2016). eprint: https://www.biorxiv.org/content/early/2016/11/14/085050.full.pdf.

[12] Murray Patterson, Tobias Marschall, Nadia Pisanti, Leo van Iersel, Leen Stougie, Gunnar W. Klau, and Alexander Schönhuth. “WhatsHap: Weighted Haplotype Assembly for Future-Generation Sequencing Reads”. In: *Journal of Computational Biology* 22.6 (2015). PMID: 25658651, pp. 498–509. eprint: https://doi.org/10.1089/cmb.2014.0157.

[13] Ryan Tewhey, Vikas Bansal, Ali Torkamani, Eric Topol, and Nicholas Schork. “The importance of phase information for human genomics”. In: *Nature reviews. Genetics* 12 (Mar. 2011), pp. 215–23.

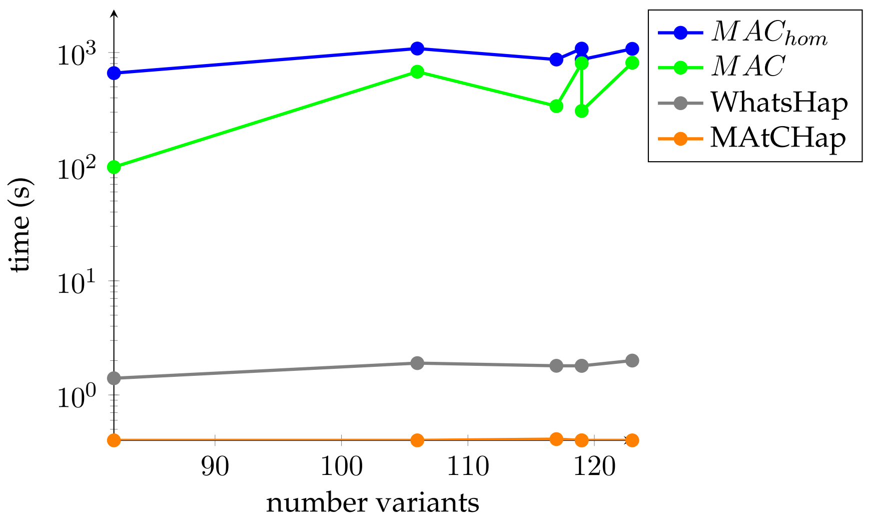
[14] Rui-Sheng Wang, Ling-Yun Wu, Zhen-Ping Li, and Xiang-Sun Zhang. "Haplotype reconstruction from SNP fragments by minimum error correction". In: *Bioinformatics* 21.10 (Feb. 2005), pp. 2456–2462. eprint: https://academic.oup.com/bioinformatics/article-pdf/21/10/2456/543733/bti352.pdf.

[15] Justin Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, Ziming Weng, Yuling Liu, Christopher Mason, Noah Alexander, Elizabeth Henaff, Alexa McIntyre, Dhruva Chandramohan, Feng Chen, Erich Jaeger, Ali Moshrefi, Khoa Pham, William Stedman, Tiffany Liang, and Marc Salit. "Extensive sequencing of seven human genomes to characterize benchmark reference materials". In: *Scientific Data* 3 (June 2016), p. 160025.

A Source code

The following link was checked on March 11, 2020.
The source code and results of this bachelor thesis:
https://gitlab.cs.uni-duesseldorf.de/schrinner/bsc-thesis-maximum-cooccurence-phasing

B Results Chromosome 2



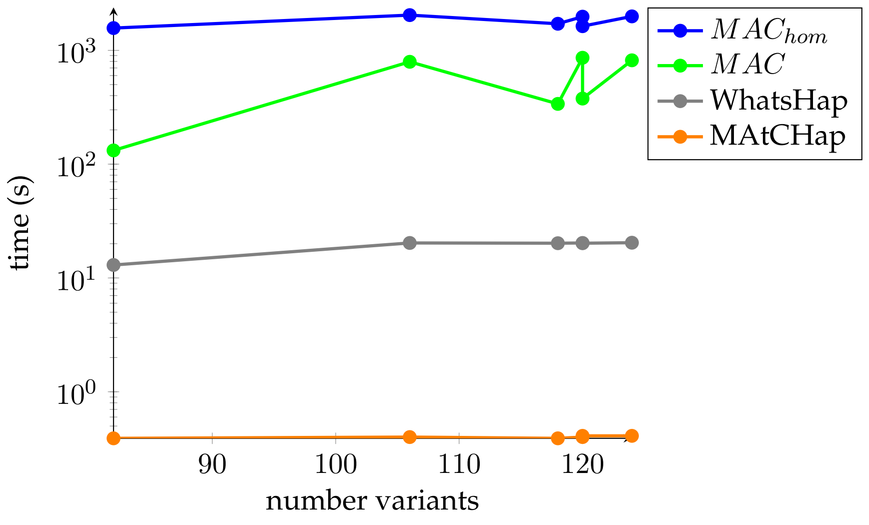
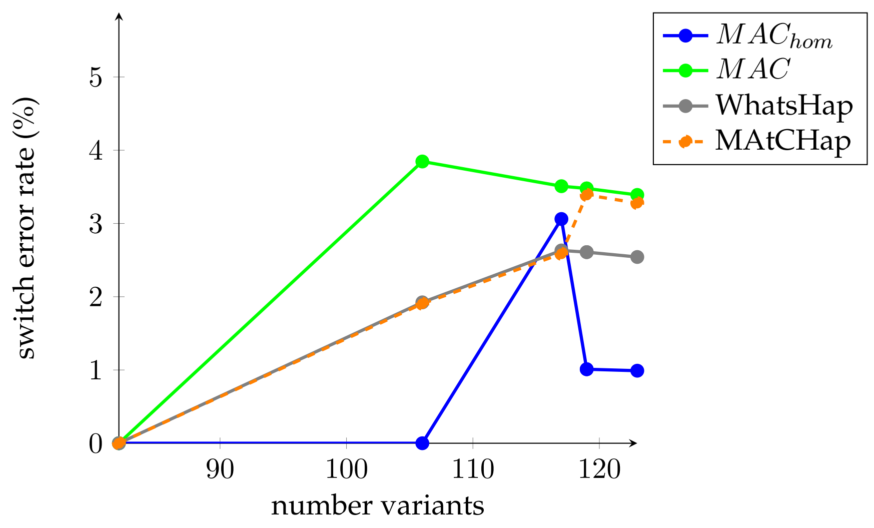


Figure 7: These plots show the running times of WhatsHap, MAtCHap and the implementations of MAC and MAC_{hom} measured for different regions on chromosome two for coverage 10× (top) and coverage 15× (bottom).



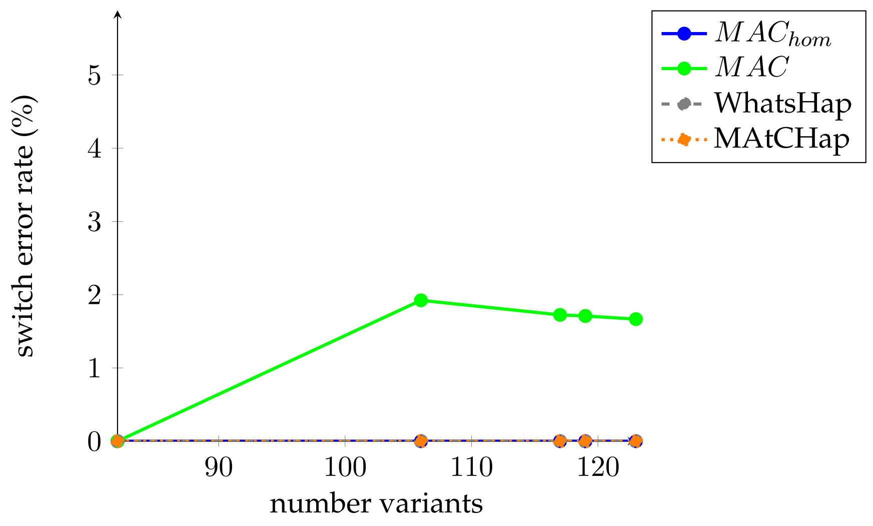
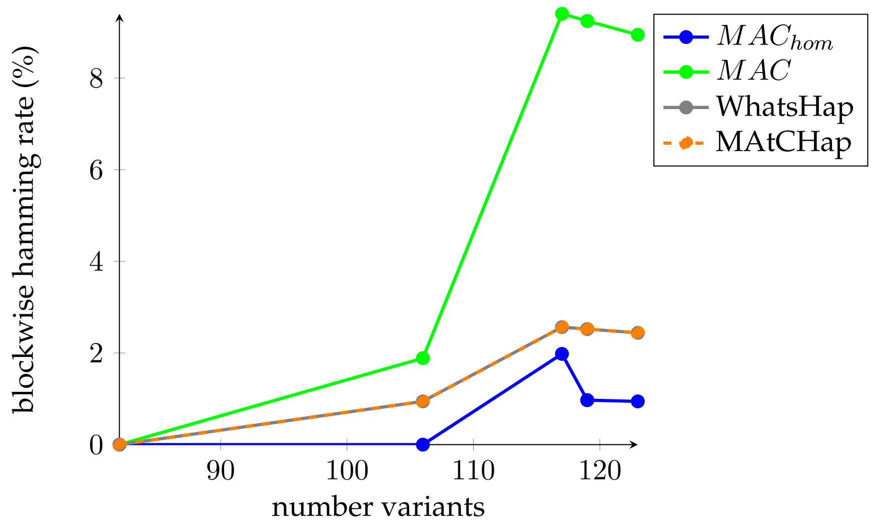


Figure 8: These plots show the switch error rates of WhatsHap, MAtCHap and the implementations of MAC and MAC_{hom} measured for phasings of different regions on chromosome two for coverage 10$\times$ (top) and coverage 15$\times$ (bottom).



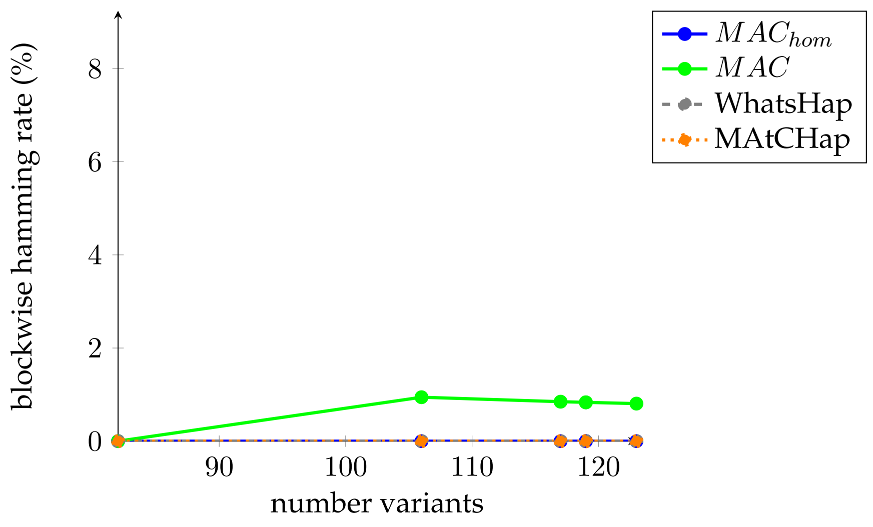


Figure 9: These plots show the block-wise Hamming distance rates of WhatsHap, MAtCHap and the implementations of MAC and MAC_{hom} measured for phasings of different regions on chromosome two for coverage 10× (top) and coverage 15× (bottom).