Saarland University
Center for Bioinformatics
Bachelor's Program in Bioinformatics

Bachelor's Thesis in Bioinformatics

Haplotype Phasing from Hi-C data

submitted by

Andrea Kupitz

on September 2017

Supervisor
Professor Dr. Tobias Marschall

Advisor
Shilpa Garg

Reviewers
Professor Dr. Tobias Marschall
reviewer2

Zentrum für Bioinformatik

Kupitz, Andrea
Haplotype Phasing from Hi-C data
Bachelor's Thesis in Bioinformatics
Universität des Saarlandes
Saarbrücken, Germany
September 2017

Declaration

I hereby confirm that this thesis is my own work and that I have documented all sources used.

I hereby declare that the submitted digital and hardcopy versions of this thesis correspond to each other. I give permission to the Universität des Saarlandes to duplicate and publish this work.

Saarbrücken, on September 2017

Andrea Kupitz

Abstract

todo

Acknowledgments

I would like to thank deeply Prof. Dr. Tobias Marschall and Shilpa Garg, who introduced me to my bachelor theme and always offered me their advice and support which I appreciated very much.

I owe thanks to both of them for the opportunity to develop this bachelor thesis at the *Center for Bioinformatics*.

Furthermore I would like to thank all other people who put in touch with me during the development of this bachelor thesis and who were very helpful in providing information.

Contents

List of Tables

List of Figures

List of Abbreviations

DNA deoxyribonucleic acid 2
MEC minimum error correction 3
SNP single nucleotide polymorphism 1
wMEC weighted minimum error correction 4
dNTP deoxynucleotide 2
VCF variant coverage format 7

Chapter 1

Introduction

1.1 Haplotype Phasing

As the human genome is diploid, it consists of two sets of genetic information; one is derived from the mother and one from the father. The inherited chromosome sets differ from each other in only a few heterozygous nucleotide positions, so-called single nucleotide polymorphisms (SNPs). Each parental list of SNPs is defined as a *haplotype*. In this way, haplotypes reveal the affiliation of genetic variants to the sets of chromosomes. In contrast to that, the genotype only distinguishes between homo- and heterozygous positions, and one cannot determine if multiple variants arise from the same origin. Knowing if two genes belong to the same haplotype, is relevant for inheritance of genetic deseases and thus also for diagnostic analysis. [1][2]
There exist different basic approaches for *haplotype phasing*, the determination of haplotypes. The first method, *molecular phasing*, physically separates sequencing reads in molecules before further analysing them. Snyder et al. introduced two methods for molecular phasing in their paper from 2015. They differ between dense methods where many smaller blocks are phased and sparse methods, where up to whole chromosomes may be directly phased. [3]
Whereas *population-based phasing* assigns genomic regions to haplotypes by using the most frequent allele from reference data. For example, the approach of Stephens, Smith and Donnelly(2001) constructs a Marcov chain of haplotypes from known genotypes to obtain the sample haplotype. [4]
Genetic phasing uses genotype information from closely related individuals for haplotyping. Roach et al.(2011) create pedigrees from whole families in order to determine the inheritance of every variant. [5]
Finally, the method to be considered in the following, *haplotype assembly* or *read-based phasing* takes sequencing reads into account for an algorithmic haplotype assembly. This technique is an only computational approach with no need for genetic information of other individuals or usage of any other biological methods besides sequencing technologies. [2]
But, using haplotype phasing, there exist regions which are uncovered by phasing information, so-called variant deserts. Some new approaches try to cover yet uncovered regions using *Hi-C* reads; one of them will be introduced in the following section.

1.2 Hi-C Method

Hi-C is a technique to capture existing interactions between chromosomal fragments with each other. These interactions exist because the deoxyribonucleic acid (DNA) is closely packaged together in the nucleus of a cell. Thereby different parts of DNA overlap, which are nearby in three-dimensional space but may not be located on the same strand. It is reasonable to assume, that interactions occur more frequently between fragments next to each other on DNA-strands. Moreover, the conformation of the chromatin is highly related to its biological functionality. For example, promotor and enhancer regions tend to be more tightly packed together. [6]
In order to receive the sequence of interacting regions, one needs to perform several molecular biological methods. This stategy was previously described and realised by van Berkum et al. in 2010. First one should crosslink adjacent chromatin segments using formaldehyde. Then the strands are cut with the restriction enzyme *HindIII*, such that one can isolate the crosslinked parts from the remaining chromatin. The resulting restriction overhangs get filled up with biotin-labeled deoxynucleotides (dNTPs). Afterwards, blunt-end ligation is performed under extremely dilute conditions in order to favor ligation events between crosslinked fragments. During ligation the recognition sequence for the previously used restriction enzyme is lost, and a NheI restriction site is created, which may be used for testing the efficiency of ligation and marking in the end. The next steps will prepare the fragments for sequencing. By reverting crosslinking and removing biotin from unligated strands, purification of DNA ensued. For sequencing, the fragments need to be sheared to a size of 300 to 500 basepairs. After shearing and repairing the sheared ends, the biotin-labeled DNA is pulled down: Streptavidin beads bind to the labeled fragments with the result that one may expect only Hi-C DNA to be remaining. Finally, Illumina paired-end sequencing is performed and the reads are aligned to a reference genome to determine their native location on the strands. [7]
Reads arise from different sites of the strands which were linked, but are apart from each other. Hence, a paired-end read consist of two reads which are connected by a gap and no sequence information is available between such a so-called insert. So, a special characteristic of Hi-C reads is, that these small reads may span large distances in the genome if they were located far away from each other in the genome and thus bridge uncovered regions during assembly. [8]

1.3 Related Work

Because of this ability of Hi-C reads to bridge regions where no phase information is available, there already exist approaches which assemble haplotypes using Hi-C data. One algorithm working with Hi-C data is *HapCut2*. *HapCut2* is an extension of the *HapCut* algorithm. [8]
HapCut was introduced by Bansal and Bafna in 2008 as an efficient and accurate algorithm for haplotype phasing. In contrast to HapCut2, it is not able to process Hi-C data. Both algorithms assemble haplotypes by computing a bipartition of the input sequencing readset. The Bipartitioning is performed by solving the *minimum error correction* (MEC) problem in the following way. Each read should either match haplotype one or haplotype two. But, in most of

the cases, the read cannot be mapped to one haplotype without any conflicts. A conflict is a position where a nucleotide of a read differs from the one at the same position in the haplotype. The goal is to create a bipartition in a way that a minimal number of conflicting positions have to be changed in order to assemble all reads to the haplotype pair. The problem of finding this bipartition is defined as minimum error correction (MEC) problem. [8]
HapCut performs the following steps iteratively until no more improvements on the number of error corrections can be made. First a read-haplotype graph is constructed. Each vertex represents one heterozygous variant, which is covered by at least one read. An edge is drawn if a pair of variants is linked by a fragment which maps the current haplotype. Now starting with a random selected haplotype, a cut in the graph is computed by flipping a set of positions to another haplotype. If the number of error corrections is lower for the resulting haplotype pair than for the previous one, the old haplotype is discarded and the current haplotype graph is used to compute a new cut. [9]
HapCut2, developed by Edge, Bafna and Bansal in 2016, is based on the same idea, but includes several optimizations regarding runtime and memory usage. In order to reduce the amount of needed space, only edges between adjacent variants are drawn in the haplotype graph. Furthermore, instead of using costs for error correction, a heuristic likelihood-based model is provided. For a given probability q that a variant i in a read R is correct, the likelihood of observing a haplotype H is calculated as:

$$p(R_i|q,H) = \prod_{j,R_i[j]\neq -} \delta(R_i[j],H[j])(1-q_i[j]) + (1-\delta(R_i[j],H[j]))q_i[j]. \quad (1.1)$$

The probabilities are adjusted to the special characteristics of different sequencing technologies and capture their sequencing errors. [8]
In this way, the algorithm is well designed for special assembling issues, which may occur with Hi-C data. For instance, the very variable size of Hi-C reads, depending on the insert size between the two sequenced blocks. But HapCut2 also considers the number of h-trans errors, which are related to the insert sizes. These errors in data originate from bad ligation events between fragments from opposite homologous chromosomes. [8]
But this haplotype assembly approach also faces some limitations. As already mentioned, HapCut2 implements methods to reduce the time spend during computation. Nevertheless, the algorithm needs to consider all pairs of edges per fragment. So, the runtime of calculating the likelihoods is quadratically in the mximum number of variants per read, linearly in the number of all variants and also linearly in the average coverage per variant. And the runtime of updating the likelihoods after a cut was found depends linearithmic on the number of variants [8]
In contrast to that, here we provide *WhatsHap*, an algorithm for haplotype phasing, which is independent of read length. [10]

1.4 WhatsHap

WhatsHap is a haplotype assembly tool which takes long, next generation sequencing reads and assembles them to two haplotypes by solving the *weighted minimum error correction* (wMEC) Problem. As WhatsHap may also take

pedegree information into account, it is able to combine read-based and genetic phasing. [**Shilpa**]
By creating a bipartition of the readset, respectively one haplotype is assembled from each partition. Bipartitioning starts with all reads covering the leftmost position. Assingning the first position is trivial without sequencing errors. Each reads first position belongs either to the reference or to the alternate allele or is marked as a gap if it does not match any. Contineously read positions are assigned to haplotypes, resulting in a bipartition of the readset. As soon as sequencing errors occur, there will be conflicting positions, where nucleotides in reads do not match any of the ones in the assembled haplotype at the same position. The task to find a solution with a minimum number of error corrections is called a weighted minimum error correction (wMEC) problem. Here the weighted read positions correspond to the probabilities that the position is correctly sequenced. [10]
In this way, the weight W of setting a set of reads R at position j to 0 is defined as the sum of weights w of setting each fragment F in the readset to 0:

$$W^0(j,R) = \sum_{\substack{i \in R \\ \mathcal{F}(i,j)}} w(i,j). \tag{1.2}$$

The same holds true for setting all positions to 1. And thus the cost C of assigning a bipartition (R,S) is

$$\Delta_C(j,(R,S) := minW^0(j,R), W^1(j,R) + minW^0(j,S), W^1(j,S). \tag{1.3}$$

Recursively one may compute the costs of the following position $C(j+1,\cdot)$ from $C(j,\cdot)$ as

$$C(j+1,(R,S)) = \Delta_C(j+1,(R,S)) + \min_{B \in \mathcal{B}(F(j)|R^{\cap}_{j+1}, S^{\cap}_{j+1}))} C(j,B). \tag{1.4}$$

An optimal bipartition can be obtained by backtracking the created dynamic programming table $\overline{D}(j)$:

$$\overline{D}(j,B) = \operatorname*{argmin}_{B' \in \mathcal{B}(F(j)|B)} C(j,B'). \tag{1.5}$$

The algorithm works especially well for long reads because its runtime is independant of the length of reads. But rather it depends linearly on the number of SNP positions and exponentially on the read coverage. In this way, WhatsHap is able to handle de novo sequencing reads of increasing length. Moreover, the user is able to set the maximal coverage to reduce the algorithms runtime. By default this parameter is set to 15 reads per variant. It was previously shown in paper by Patterson et al., that this size is sufficient to produce feasible results. [10]
To reduce the coverage in order to keep the runtime low, a read selection preprocessing step is performed. The results using this heuristic instead of a random read selection were more efficient. [11]

1.4.1 Read Selection

Read selection is a heuristic approach which selects reads iteratively in two steps until no more reads can be selected. The goal is to maximize the coverage of all

variants by selecting the minimum number of reads. So, on the one hand one aims to use variants which are well-connected by high-quality reads but on the other hand needs to stay under the coverage threshold. [11]

Before selecting the first reads, all reads which cover less than two variants are discarded, because one gains no phase information of them. Then the algorithm starts to select disjoint slices of reads. This implies, that every read, which was included in a previous slice, is never selected again. With every slice one tries to cover all variants once, selecting reads greedily, starting with the one with the best score. Three values are computed to score reads. [11]
At first reads are characterized by the number of variants which are covered by a read R. One can compute this value as:

$$score_{static}(R) := |variants(R)| - |holes(R)|,$$

where $variants(R)$ is the number of variants that are covered by R and $holes(R)$ is the number of uncovered variants between covered variants. This score is called *static score* because it does not change during read selection. Whereas

$$score_{dyn}(R) := |variants(R)| - |variants(R) \cap variants(\mathcal{R}_S)| - |holes(R)|$$

changes as soon as the set of selected reads $\mathcal{R}_S$ changes. If two reads have the same $score_{static}$ and $score_{dyn}$, the $score_{qual}$ is considered. This is defined as:

$$score_{qual}(R) := \min_{\sqsubseteq \in varints(R)} quality(R, V)$$

and read quality $quality(R, V)$ is determined by the variant V of the read with the worst mapping and base quality. [11]
In the *selectslice* step a read is added to a slice if it spans yet uncovered variants and it does not violate the coverage constraint. The dynamic scores are updated after every time the slice changed. [11]

After a slice got filled up as far as possible, the algorithm tries to cover uncovered variants in a bridging step. Therefore, connected components within the reads in a slice are detected. A component is composed of multiple reads that are linked with each other by overlaps. These overlaps result from variants which are covered by more than one read. An union-find data structure is used in order to find connected components, and reads are selected, starting from the one with the best score. The algorithm continues selecting bridging reads until there are no more candidates left. Afterwards, reads which got connected by bridging reads, are merged together to one component. In this way, the coverage of overlapping positions gets reduced and leaves more space for the next iteration. Each iteration starts building a new slice by performing the two steps of read selection again until all reads, which do not violate the coverage constraint, belong to a slice. [11]

1.4.2 Limitations

Read selection works well for single-end reads. Even if one wants to use reads with higher error rates or select a lower maximal read coverage of five or ten times, the results stay accurate. [12] Contrary to that, for paired-end reads

the quality of the phasing could be improved. For Illumina reads the number of variants which were covered by reads for phasing is about the half of the achievable in the best case.
The bad coverage of variant positions are based of the haplotype phasing strategy. The algorithm calculates the best bipartition of reads, starting from the leftmost position. And a final decision for the related haplotype of every read is made once the rightmost position of a read was seen. Therefore, gaps without phase information also account to the coverage. This also applies for the read selection. [10]
As already mentioned, Hi-C reads may contain large gaps between the mate pairs. These gap positions also increase the coverage, even though variants covered by gaps do not provide phase information. As a consequence, during read selection, the maximum coverage is reached in the selection step and no more bridging reads can be selected. The absence of the bridging step leads to many unconnected read blocks and thus the coverage does not decrease.

1.4.3 Aims

Because the current implementation of read selection faces some limitations using Hi-C data, we want to offer a new algorithmic solution for haplotype assembly here.
Hi-C reads consists of two small reads which may be separated by large inserts. Therefore, they are able to connect yet unconnected regions in the genome. But also because of these special characteristics, the read selection is inefficient for Hi-C data because the inserts also account to the coverage.
Here we aim to present new approaches for an efficient, fast and accurate haplotype phasing using Hi-C data. Our solution should work fast for long reads, and additionally we want to overcome the current limitations using Hi-C reads.

Chapter 2

New Read Selection

2.1 Iterative WhatsHap

The first approach to improve the results of WhatsHap using Hi-C data was to run WhatsHap iteratively. WhatsHap enables the user to an iterative usage because it outputs a phased variant coverage format (VCF) file. This phased VCF file may be part of the input of the next iteration besides a VCF and a binary alignment/map (BAM) file, which are used for every iteration. A VCF file contains known hetero- and homozygous positions in the genome, whereas a phased file provides information of phased variants. Another input was changed for every iteration, the BAM files. These binary compressed files comprise sequence alignments in the form of mapped reads.
As the main challenge of Hi-C reads are the large inserts, the idea was to consider different insert sizes and their effect on the phasing quality.

2.2 Longest Path in Graph Algorithm

Chapter 3

Evaluation

Chapter 4

Discussion

todo

Appendix A

Regulation

Bibliography

[1] R. Tewhey et al. "The importance of phase information for human genomics". In: *Nature* 12 (2011), pp. 215–223. DOI: 10.1038/nrg2950.

[2] G. Glusmann et al. "Whole-genome haplotyping approaches and genomic medicine". In: *Genome Medicine* 73.6 (2014). DOI: 10.1186/s13073-014-0073-7.

[3] M. W. Snyder et al. "Haplotype-resolved genome sequencing: experimental methods and applications". In: *Nature Reviews Genetics* 16 (2015), pp. 344–358. DOI: 10.1038/nrg3903.

[4] M. Stephens et al. "A New Statistical Method for Haplotype Reconstruction from Population Data". In: *Am J Hum Genet* 68.4 (2001), pp. 978–989. DOI: 10.1086/319501.

[5] C. J. Roach et al. "Chromosomal Haplotypes by Genetic Phasing of Human Families". In: *AJHG* 89.3 (2011), pp. 382–397. DOI: 10.1016/j.ajhg.2011.07.023.

[6] J.-M. Belton et al. "Hi-C: A comprehensive technique to capture the conformation of genomes". In: *Methods.* 58.3 (2012), pp. 268–276. DOI: 10.1016/j.ymeth.2012.05.001.

[7] N. L. van Berkum et al. "Hi-C: A Method to Study the Three-dimensional Architecture of Genomes." In: *J VIs Exp* 1869.39 (2010). DOI: 10.3791/1869.

[8] P. Edge et al. "HapCUT2: robust and accurate haplotype assembly for diverse sequencing technologies". In: *Genome Res.* (2016). DOI: 10.1101/gr.213462.116.

[9] V. Bansal and V. Bafna. "HapCUT: an efficient and accurate algorithm for the haplotype assembly problem". In: *Bioinformatics* 24.16 (2008), pp. 153–159. DOI: 10.1093/bioinformatics/btn298.

[10] M. Patterson et al. "WhatsHap: Weighted Haplotype Assembly for Future-Generation Sequencing Reads". In: *Journal of Comutational Biology* 22.6 (2015), pp. 498–509. DOI: 10.1089/cmb.2014.0157.

[11] S. O. Fischer and T. Marschall. "Selecting Reads for Haplotype Assembly". In: *bioRxiv* (2016). DOI: 10.1101/046771.

[12] M. Martin et al. "WhatsHap: fast and accurate read-based phasing". In: *bioRxiv* (2016). DOI: 10.1101/085050.