HapCHAT: Adaptive haplotype assembly for efficiently leveraging high coverage in long reads

Stefano Beretta[1,†], Murray Patterson[1,†,*], Simone Zaccaria[1,2], Gianluca Della Vedova[1], and Paola Bonizzoni[1,*]

[1] Department of Informatics, Systems, and Communication, University of Milano-Bicocca, Milano, Italy and
[2] Department of Computer Science, Princeton University, Princeton, NJ, US.

[†] The authors wish it to be known that, in their opinion, the first two authors should be regarded as joint First Authors
∗ To whom correspondence should be addressed.

Abstract

Motivation: Haplotype assembly is the process of reconstructing the haplotypes of an individual from sequencing reads. Computational methods for this problem have shown to achieve high accuracy on long reads, which are becoming cheaper to produce and more widely available. Larger amounts of data, usually originating from increased coverage, are highly beneficial for improving the quality of the detection of the genetic variations that are intrinsic to the diploid nature of the human genome. However, the high accuracy of such methods comes at a cost of computational resources. The increased error rates that affect all current long-read technologies require even higher coverage: making the analysis of such data the key computational task to be solved in order to improve the accuracy of the predictions made by haplotype assembly methods.
Results: We propose a new computational approach for assembling haplotypes that is specifically designed to cope with a different error rate at each variant site, while minimizing the total number of these corrections necessary for a feasible solution. The complete strategy has been implemented in HapCHAT: Haplotype Assembly Coverage Handling by Adapting Thresholds. An experimental analysis on sequencing reads with up to 60× coverage reveals accuracy improvements with increasing coverage. Moreover, despite the fact that this adaptive approach is slightly heuristic, the extent of its efficiency over current state-of-the-art long-read haplotype assembly

methods allows the leveraging of higher enough coverage that it pays of in terms of accuracy prediction, while decreasing the running time.

Keywords: Single Individual Haplotyping, Long Reads, Haplotype Assembly, Minimum Error Correction,
Availability: HapCHAT is available at https://hapchat.algolab.eu under the GPL license.
Contact: murray.patterson@unimib.it, bonizzoni@disco.unimib.it

1 Introduction

Due to the diploid nature of the human genome, knowing the individual haplotypes on which a particular variation or set of variations occurs has a strong impact on various studies in genetics, from population genomics (Browning and Browning, 2011; Tewhey *et al.*, 2011), to clinical and medical genetics Glusman *et al.* (2014), or to the effects of compound heterozygosity (Tewhey *et al.*, 2011; Roach *et al.*, 2010), such as cancer (van de Ven *et al.*, 2012). The reconstruction of the haplotypes of the human genome of an individual from sequencing reads, known as haplotype assembly or read-based phasing, is made possible either by sequencing technologies supported by experimentally (molecular) phasing (Amini *et al.*, 2014; Snyder *et al.*, 2015), or computational methods. While experimental phasing suffers from several technological limitations that still make it a very expensive solution, computational methods continue to thrive as sequencing reads continue to become larger, of higher quality, and more widely available.

Due to the availability of curated, high-quality haplotype reference panels on a large population of individuals (Loh *et al.*, 2016; O'Connell *et al.*, 2016), the state of the art in computational methods for inferring the haplotypes of an individual are population-based statistical methods that infer a maximum-likelihood phasing (Li and Stephens, 2003; Browning and Browning, 2011), given these panels. The accuracy of these methods, however, depends heavily on the size and diversity of the population used to compile these panels, meaning poor performance on rare variants, while *de novo* variants are completely missed. These types of variants appear in the sequencing reads of the individual, making haplotype assembly (or read-based phasing) the obvious solution, since it involves the assembly of a pair of haplotypes directly from the reads. However, since it is necessary to have reads spanning enough pairs of neighboring variants to make this type of approach effective, second generation reads have been historically too short to allow haplotype assembly

to gain any momentum. However, the recent improvements in sequencing of long reads encourages the use of haplotype assembly, which may offer cost-effective methods that support accurate and comprehensive haplotype reconstruction. Current third generation sequencing platforms offered by Pacific Biosciences (PacBio) and Oxford Nanopore Technologies (ONT) are able to produce much longer reads than the previous generations, where reads are now several megabases in length, and are hence much more capable to capture together more variants than the short reads that are commonplace today (Roberts *et al.*, 2013; Kuleshov *et al.*, 2014; Ip *et al.*, 2015). Nanopore technologies still have limitations for read-phasing due to a high error-rate while PacBio technologies offer long reads with an acceptable error rate making it promising for accurate read-based phasing by the intensive use of computational methods.

Read-based phasing is closely tied to the *Minimum Error Correction* (MEC) Problem, since solving the MEC is to find an assignment of the sequencing reads to a pair of haplotypes (*i.e.*, a phasing) that minimizes the number of errors to correct. Since this task is NP-hard (Lippert *et al.*, 2002), methods for haplotype assembly have been historically either heuristic (Bansal and Bafna, 2008; Mazrouee and Wang, 2014), while exact methods are *Integer Linear Programming* (ILP) approaches (Fouilhoux and Mahjoub, 2012; Chen *et al.*, 2013), or *Dynamic Programming* (DP) approaches that are *Fixed-Parameter Tractible* (FPT) in some parameter (He *et al.*, 2010). The success of these exact methods has been, rather ironically, based on the fact that reads are short, however, since the ILP approach of (Chen *et al.*, 2013) works best on short blocks, and has preprocessing steps to ensure this, while the DP approach of (He *et al.*, 2010) has read-length as the fixed parameter. Both of these cases rely on the reads being short, and have been shown to scale poorly, or not at all, for long-read data, even when the coverage is low (Patterson *et al.*, 2014). The need for taking advantage of long reads with the advent of the third generation technologies such as PacBio and ONT, along with the deficiencies of the above methods has brought about a new generation of haplotype assembly methods targeting long reads (Deng *et al.*, 2013; Kuleshov, 2014; Patterson *et al.*, 2014, 2015) — these methods are FPT in the *coverage*, rather than the read-length, and (Patterson *et al.*, 2014, 2015), is actually *linear* in the read-length, allowing it to scale to any read-length that is available.

These methods that are FPT, *i.e.*, *exponential*, in the coverage, have moved the problem from read-length to the coverage. While this has been a positive move in general — this new wave of methods can tease out a better overall accuracy from sequencing reads than its predecessors — these

data still have a high coverage, which remains an important parameter. Hence, there is still much work to be done in order to tackle high coverage: WhatsHap (Patterson *et al.*, 2014, 2015), for example, cannot handle a maximum coverage much higher than 20×, while current datasets have average coverages well over 60× (Zook, 2014). This problem has been partially addressed with HapCol (Pirola *et al.*, 2015), a method that is exponential in the number of errors to correct at each site, since this is in general much lower than the coverage at a site, given that error rates of PacBio tend to be quite reasonable (around 15% (Roberts *et al.*, 2013)). This allows scaling to 25–30× coverage over WhatsHap at 20×, while using less time, and significantly less memory (Pirola *et al.*, 2015).

In this work, we handle even higher coverage than HapCol (Pirola *et al.*, 2015) by allowing the adjustment of the number of errors to correct at each site, dynamically, as the method runs (*cf.* Section 3.2). This allows both the handling of columns that require more errors than average (for which HapCol fails), while not exploring scenarios that involve a number of corrections that is much higher than necessary for a site, saving a significant amount of time. This is coupled with a merging procedure (*cf.* Section 3.1) that merges pairs of reads that are highly likely to originate from the same haplotype, allowing this method to scale to the coverages of today. The complete strategy has been implemented in HapCHAT: Haplotype Assembly Coverage Handling by Adapting Thresholds. An experimental analysis on real and simulated sequencing reads with up to 60× coverage reveals that we are able to leverage high coverage towards better accuracy predictions, as we see a trend towards increasing accuracy as coverage increases. We compare our method to some of the state-of-the-art methods in haplotype assembly: HapCol (Pirola *et al.*, 2015); the newest version of WhatsHap (Martin *et al.*, 2016), to which many features have since been added; and HapCUT2 (Edge *et al.*, 2016), which has also improved significantly over its predecessor, HapCUT (Bansal and Bafna, 2008). We show, not only that HapCHAT is more efficient than WhatsHap and HapCol, but that it also obtains better accuracy predictions than these methods, indeed by better leveraging the information provided in the high coverage of these sequencing reads.

2 Background

Let v be a vector, then $v[i]$ denotes the value of v at position i. A *haplotype* is a vector $h \in \{0,1\}^m$. Given two haplotypes of an individual, say h_1, h_2, the position j is *heterozygous* if $h_1[j] \neq h_2[j]$, otherwise j is *homozygous*. A

fragment is a vector f of length l belonging to $\{0,1,-\}^l$. Given a fragment f, position j is a *hole* if $f[j] = -$, while a *gap* is a maximal sub-vector of f of holes that is, a gap is preceded and followed by a non-hole element (or by a boundary of the fragment).

A *fragment matrix* is a matrix M that consists of n rows (fragments) and m columns (SNPs). We denote as L the maximum length for all the fragments in M, and as M_j the j-th column of M. Notice that each column of M is a vector in $\{0,1,-\}^n$ while each row is a vector in $\{0,1,-\}^m$.

Given two row vectors r_1 and r_2 belonging to $\{0,1,-\}^m$, r_1 and r_2 are in *conflict* if there exists a position j, with $1 \leq j \leq m$, such that $r_1[j] \neq r_2[j]$ and $r_1[j], r_2[j] \neq -$, otherwise r_1 and r_2 are in *agreement*. A fragment matrix M is *conflict free* if and only if there exist two haplotypes h_1, h_2 such that each row of M is in agreement with one of h_1 and h_2. Equivalently, M is conflict free if and only if there exists a *bipartition* (P_1, P_2) of the fragments in M such that each pair of fragments in P_1 is in agreement and each pair of fragments in P_2 is in agreement. A *k-correction* of a column M_j, is obtained from M_j by flipping at most k values that are different from $-$. A column of a matrix is called *homozygous* if it contains no 0 or no 1, otherwise (if it contains both 0 and 1) it is called *heterozygous*. We say that a fragment i is *active* on a column M_j, if $M_j[i] = 0$ or $M_j[i] = 1$. The *active fragments* of a column M_j are the set $active(M_j) = \{i : M_j[i] \neq -\}$. The *coverage* of the column M_j is defined as the number cov_j of fragments that are active on M_j, that is $cov_j = |active(M_j)|$. In the following, we indicate as cov the maximum coverage over all the columns of M. Given two columns M_i and M_j, we denote by $active(M_i, M_j)$ the intersection $active(M_i) \cap active(M_j)$. Moreover, we will write $M_i \approx M_j$, and say that M_i, M_j are in accordance, if $M_i[r] = M_y[r]$ for each $r \in active(M_i, M_j)$, or $M_i[r] \neq M_y[r]$ for each $r \in active(M_i, M_j)$. Notice that $M_i \approx M_j$ means that these two columns are compatible, that is, they induce no conflict. Moreover, $d(M_i, M_j)$ denotes the minimum number of corrections to make columns M_i and M_j in accordance.

The *Minimum Error Correction* (MEC) problem (Lippert *et al.*, 2002; Bonizzoni *et al.*, 2003), given a matrix M of fragments, asks to find a conflict free matrix C obtained from M with the minimum number of corrections. In this work, we consider the variant of the MEC problem, called k-cMEC in which the number of corrections per column is bounded by an integer k (Pirola *et al.*, 2015). More precisely, we want a *k-correction matrix* D for M where each column C_j is a k-correction of column M_j, minimizing the total number of corrections. We recall that in this paper we will consider only matrices where all columns are heterozygous.

Now, let us briefly recall the dynamic programming approach to solve the

k-cMEC problem (Pirola *et al.*, 2015). This approach computes a bidimensional array $D[j, C_j]$ for each column $j \geq 1$ and each possible heterozygous k-correction C_j of M_j, where each entry $D[j, C_j]$ contains the minimum number of corrections to obtain a k-correction matrix C for M on columns $M_1, \ldots, M_j$ such that the columns C_j are heterozygous. For the sake of simplicity, we pose $D[0, \cdot] = 0$. For $0 < j \leq m$, the recurrence equation for $D[j, C_j]$ is the following, where δ_j is the set of all heterozygous k-corrections of the column M_j.

$$D[j, C_j] = \min_{C_{j-1} \in \delta_{j-1}, C_j \approx C_{j-1}} \Big\{ D[j-1, C_{j-1}] + d(M_j, C_j) \Big\}.$$

For the complete description of the dynamic programming recurrence we refer the reader to (Pirola *et al.*, 2015; Bonizzoni *et al.*, 2015).

3 Methods

In this section, we highlight the new insight of HapCHAT for the assembly of single individual haplotypes, with the specific goal of processing high coverage datasets. In fact, as reported in the original HapCol paper (Pirola *et al.*, 2015), the FPT algorithm is exponential in the coverage and the number k of allowed corrections in each position. Therefore, we developed a pre-processing step which merges reads belonging to the same haplotype based on a graph clustering method. Moreover, we also improved the HapCol method by introducing a heuristic procedure to cope with problematic positions, i.e. those requiring more than k corrections.

As anticipated, the combination of all these improvements allowed the possibility of reconstructing haplotypes using higher coverage (w.r.t. the original HapCol method) reads, while reducing the runtimes.

3.1 Pre-processing

The first step of our pipeline is to merge pairs of fragments that, with high probability, originate from the same haplotype. With p we denote the probability that a single nucleotide is an error — we recall that $p \approx 0.15$ and that errors are uniformly distributed for PacBio reads. Let r_1 and r_2 be two reads that share $m + x$ sites, agree on m of those sites and disagree on x sites. Then the probability that r_1 and r_2 originate from the same haplotype is approximately $p_s(r_1, r_2) = p^x(1 - p/3^m)$, while the probability that they originate from two different haplotypes is approximately $p_d(r_1, r_2) = p^m(1 - p/3^x)$.

A simple approach to reduce the size of the instance is to merge all pairs (r_1, r_2) of fragments such that $p_s(r_1, r_2)$ is sufficiently large. But that would also merge some pairs of fragments whose probability p_d is too large. Since we wanted to be conservative in merging fragments, we partition the fragment set into clusters such that $p_s/p_d \geq 10^6$ for each pair of fragments in the cluster. Then, for each site, the character that is the result of a merge is chosen applying a majority rule, weighted by the Phred score of each symbol.

3.2 Adaptive k-cMEC

Here, we describe how we modified the HapCol dynamic programming recurrence in order to deal with problematic columns for which the maximum allowed number of corrections is not enough to obtain a solution. As stated in the original HapCol paper (Pirola *et al.*, 2015), the number k_j of corrections for each column M_j is computed, based on its coverage cov_j and on two input parameters: ϵ (average *error-rate*) and α (the probability that the column M_j has more than k_j errors). In fact, the number of errors varies among different columns proportional to the coverage and so HapCol estimates the number of corrections, accordingly. This is done in order to bound the value of k, which is fundamental since HapCol implements an FPT algorithm that is exponential in both coverage and maximum number of allowed corrections. For this reason, we would prefer to have small values of k_j. A side effect of this approach is that, given k_j, there might be some columns M_j for which it is not possible to find a solution.

Therefore, we developed a heuristic procedure which has the final goal of finding, for each column M_j, a value of k_j for which a solution exists. We recall that the recurrence equation governing the original dynamic programming approach considers all k_j-correction $C_j \in \delta_j$. We slightly modify the definition of k-corrections to cope with those problematic columns, by increasing the number of allowed corrections. Let $C_{j,k}$ be a k-correction of M_j with exactly k corrections, let $h_{j,0} = k_j$ and $h_{j,i} = h_{j,i-1} + \lfloor \log_2(h_{j,i-1}) + 1 \rfloor$. Then $k_j^* = \min_{i:D[j,C_{j,h_{j,i-1}}]\neq\infty} \{h_{j,i}\}$ if $i > 0$, where $D[\cdot,\cdot] \neq \infty$ means it is a feasible correction. Starting from this notation, the new set of possible corrections of column M_j is

$$\delta_j = \left\{C_{j,k} : 1 \leq k \leq k_j^*\right\}.$$

Notice that the sequence of $h_{j,i}$ is monotonically increasing with i, hence we can compute k_j^* by starting with k_j and increasing it until we are able to find a k_j^*-correction for the column M_j. The dynamic programming equation is unchanged, but our new construction of the set δ_j guarantees that we are

always able to compute a solution. Moreover, just as for HapCol, we cannot guarantee that we solve optimally the instance of the MEC problem.

One of the key points of this procedure is how we increment $h_{j,i}$, that is by adding a logarithmic quantity. This guarantees a balance between finding a small value of k_j^* and the running time needed for the computation.

4 Results

We now describe the results of our experiements. In Subsection 4.1, we describe the data that we use, or simulate. In Subsection 4.2, we detail the experiments that we set up in order to compare our tool with others. Finally in Subsection 4.3, we discuss the results of these experiments.

4.1 Data description

The Genome in a Bottle (GIAB) Consortium has released publicly available high-quality sequencing data for seven individuals, using eleven different technologies (Zook, 2014). Since our goal is to assess the performance of different single-individual haplotype phasing methods, we study Chromosomes 1 and 21 of the Ashkenazim individual NA24385. This individual is the son in a mother-father-son trio. We downloaded from GIAB the genotype variants call sets NIST_CallsIn2Technologies_05182015, a set of variants for each individual of this trio that have been called by at least two independent variant calling technologies. In order to be able to compare against methods that use reference panels or information from multiple individuals, *e.g.*, a trio, for single-individual haplotype phasing, we considered for each chromosome all bi-allelic SNPs that: (a) appear also in the 1000 Genomes reference panel `https://mathgen.stats.ox.ac.uk/impute/1000GP_Phase3.tgz`, and (b) have been called in all three individuals of the Ashkenazim trio, *i.e.*, also in the mother and the father. This resulted in 140744 SNPs for Chromosome 1, of which 48023 are heterozygous, and 26419 for Chromosome 21, of which 8941 are heterozygous. We refer to this set of SNPs for a chromosome as the set of benchmark SNPs for the chromosome – each set is in the form of a VCF file. Since the authors of (Garg *et al.*, 2016) also study GIAB sequencing reads, and have made the pipeline for collecting and generating their data publicly available at `https://bitbucket.org/whatshap/phasing-comparison-experiments/`, we use or modify parts of this pipeline to generate our data as detailed in the following.

Phasing Benchmark Because population-based statistical methods are among the most developed types of approaches, thanks to the availability of curated, high-quality reference panels, we use the output of a statistical phaser – a phasing of the benchmark SNPs – as a benchmark for comparing all of the following (read-based) phasing methods. Note that population-based and read-based phasing approaches use completely independent sources of information for phasing – another good reason for this comparison. More specifically, we use the population-based phasing tool SHAPEITv2-r837 (Delaneau, 2013) with default parameters. We provided as input to this program, the 1000 Genomes reference panel, mentioned in (a) above, the corresponding genetic map `http://www.shapeit.fr/files/genetic_map_b37.tar.gz`, and the unphased genotypes, *i.e.*, the set of benchmark SNPs of the corresponding chromosome. It is the resulting phasing that we use as a benchmark for comparing the different read-based phasing methods.

GIAB PacBio Reads One of the more recent technologies producing long reads – those which are the most informative for read-based phasing – is the Pacific Biosciences (PacBio) platform. PacBio is one of the eleven technologies on which GIAB provides sequencing reads. We hence downloaded the set of aligned PacBio reads `ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA24385_son/PacBio_MtSinai_NIST/MtSinai_blasr_bam_GRCh37/hg002_gr37_1.bam` for chromosomes 1 and 21 of the son, which have average coverages of 60.2× and 53.3× and average mapped read lengths of 8687bp and 8712bp, respectively. We then downsampled both read sets to average coverages 25×, 30×, 35×, 40×, 45× and 50×, and the read set corresponding to chromosome 1 to the additional average coverages 55× and 60×. This was done using the `DownsampleSam` subcommand of Picard Tools, which randomly downsamples a read set by selecting each read with probability p. We downsample recursively, so that each downsampled read set with a given average coverage is a subset of any downsampled read set with an average coverage higher than it.

Simulated PacBio Data Aside from the PacBio data described in the previous section, we also produce and run our experiments on a simulated read set for each chromosome. Reference panels may leave out some variants with low allele frequency – a good reason for doing read-based phasing – and statistical methods might be susceptible to systematic bias in the data. For these reasons, we complement our study with an experimental analysis on simulated reads, as follows.

We first obtain a pair of "true" haplotypes off of which we simulate reads. Given one side of the phasing, by SHAPEIT, of the variants (the set of benchmark SNPs) of the appropriate chromosome, we incorporate this side into the reference genome (hg19) by flipping the variant sites that are the alternative allele in this side of the phasing. The opposite side is done analagously. Using each true haplotype as the input, we produce a corresponding set of reads for this haplotype using PBSIM (Ono, 2013), a Pacbio-specific read simulator. We input to PBSIM the optional parameters `--depth 60` so that our simulated reads have sufficient coverage, and as `--sample-fastq` a sample of the original GIAB PacBio reads described in the previous section, so that our simulated reads have the same length and accuracy profile as the corresponding real read set. We align the resulting simulated reads to the reference genome using BWA-MEM 0.7.12-r1039 (Li, 2013) with optional parameter `-x pacbio`. Finally, this pair of aligned read sets, representing the reads coming off of each haplotype is merged using the `MergeSamFiles` subcommand of Picard Tools, obtaining the final simulated read set. In the same way as we have done with the read sets for the real chromosomes, we downsample each to average coverages 25×, 30×, 35×, 40×, 45× and 50× and the simulated read set for chromosome 1 to the additional average coverages 55× and 60×.

To summarize, the data we use or simulate regards both real and simulated reads on chromosome 1 for a set of 8 average coverages, and on chromosome 21 for a set of 6 average coverages, for a total of 28 read sets, each in the form of a BAM file. It is on these 28 read sets, along with the the set of benchmark SNPs for each chromosome – a pair of VCF files – that we carry out our experiments, as described in the following section.

4.2 Experimental Setup

We compare our tool, HapCHAT to the most recent state-of-the-art read-based phasing methods of WhatsHap (Patterson *et al.*, 2015; Martin *et al.*, 2016), HapCol (Pirola *et al.*, 2015) and HapCUT2 (Edge *et al.*, 2016). A short description of each tool, and how we set it up for this comparison on the data described in Subsection 4.1 is as follows.

WhatsHap WhatsHap (Patterson *et al.*, 2015) is a C++ implementation of a dynamic programming algorithm for the wMEC that is *fixed parameter tractable* in the maximum coverage at any site of the input. For WhatsHap to be useful in practice, it therefore needs to prune any instance to a *target*

maximum coverage, since its runtime and memory scale exponentially with the maximum coverage at any site.

While WhatsHap has shown to outperform a selection of state-of-the art read-based phasing tools in (Patterson *et al.*, 2015), some tools have since been developed (such as HapCol (Pirola *et al.*, 2015), which we compare here, that outperform WhatsHap in terms of time and memory usage. However, WhatsHap has been enriched with many features, such as a read selection procedure, detailed in (Fischer and Marschall, 2016), which is now built-in: reads are selected down to maximum coverage 15 by default. Another feature is a *realignment* preprocessing step (Martin *et al.*, 2016) which realigns locally the variants to the alternative allele (of the reference genome), to correct some of the errors in the reads, which has been shown to significantly boost prediction accuracy (Martin *et al.*, 2016), especially for noisy reads such as PacBio. In addition to these improvements, WhatsHap is now a production-quality tool that works with standard formats such as BAM and VCF. As far as we know, it is the only production-quality tool for read-based phasing, aside from the `ReadBackedPhasing` command of GATK (DePristo *et al.*, 2011), which has been discontinued.

For each of the 28 read sets, we provide to WhatsHap (version 0.13) its BAM file and the VCF file for the corresponding chromosome. We run WhatsHap on this input pair in realignment mode by providing it with optional parameter `--reference`, the reference genome (hg19), producing the resulting phased VCF file. We then do another parallel set of runs of WhatsHap in realignment mode, but pruning to target maximum coverage 20 instead (note that the first set was pruned to the default maximum coverage of 15), by providing the optional parameter `-H 20`. It is the resulting set of 56 phasings by WhatsHap, in the form of phased VCF that we use for the basis of comparison with the other methods.

HapCol HapCol (Pirola *et al.*, 2015) is an implementation in C++ of a dynamic programming algorithm for the k-MEC problem that we explain in detail in Sections 2 and 3.2.

For each of the 28 read sets, together with the VCF file of the corresponding chromosome, we convert it to the custom input format for HapCol. Since HapCol does not have a read selection procedure – something it does need for data at 60× coverage (*cf.* Section 1) – we then apply the read selection procedure of (Fischer and Marschall, 2016) to prune this set to the target maximum coverages of 15×, 20×, 25× and 30×. On these resulting 112 input files, we run HapCol with its default value of $\alpha = 0.01$ (and of

$\varepsilon = 0.05$). Since HapCol is not adaptive, but we want to give it a chance to obtain a solution on its instance, should a given alpha be infeasible (*cf.* Section 3.2), we continue to rerun HapCol with an α of one tenth the size of the previous until a solution exists. HapCol outputs a pair of binary strings representing the phasing, which we then convert to phased VCF. It is this set of resulting 112 phasings (phased VCF files) that we use to compare with the other methods.

HapCUT2 HapCUT2 (Edge *et al.*, 2016) is heuristic for the wMEC problem which is based on finding a MAX-CUT of the corresponding graph representation of the instance. It is the successor of HapCUT (Bansal and Bafna, 2008), and has been improved to the extent it seems to be competitive enough to merit comparison.

For each of the 28 read sets, we use the `extractHAIRS` program that comes with HapCUT to convert its BAM / VCF pair into the custom input format for HapCUT2. We provide this program the optional parameter `--maxIS 600` which is the maximum insert size for a paired-end read to be considered as a single fragment for phasing (default 1000). We also extract another 28 sets of parallel inputs for HapCUT2 as above, but also providing the additional optional parameter `--indels 1`, which also extracts reads that span indels. We then ran HapCUT2 on these 56 instances, each producing a custom output which is then converted to phased VCF with the subcommand `hapcut2vcf` of the WhatsHap toolbox.

HapCHAT Our tool, HapCHAT has been partially integrated into the WhatsHap codebase, allowing us to take advantage of features such as realignment in our phasing method as well. Hence, for each of our 28 read sets, we first run HapCHAT in realignment mode followed by our merging step described in Section 3.1, which reduces the coverage. If necessary, the reads are further selected via a greedy selection approach (based on the Phred score), with ties broken at random, to downsample each dataset to the target maximum coverages of 15×, 20×, 25× and 30×.

In summary, with 28 read sets at 4 resulting target maximum coverages, we hence have 112 resulting phasings, in phased VCF format, for which the comparison of HapCHAT to other methods is based.

4.3 Experimental results

We have run all tools with a timeout of 1 day and according to the description of Section 4.2. The times reported here do not include the time necessary to

Ashkenazim		HapCHAT		HapCol		WhatsHap		HapCUT2
Chr.	Avg. Cov.	Max25	Max30	Max25	Max30	Max15	Max20	
Chr. 1	Cov. 25	**0.251**	**0.251**	0.542	-	0.254	0.254	2.359
	Cov. 30	**0.242**	**0.242**	0.489	-	0.258	0.258	2.218
	Cov. 35	0.244	**0.242**	0.466	-	0.253	0.249	2.009
	Cov. 40	0.244	**0.242**	0.459	-	0.255	0.251	1.744
	Cov. 45	**0.244**	0.246	0.426	-	0.255	0.250	1.684
	Cov. 50	0.237	**0.235**	0.404	-	0.244	**0.235**	1.539
	Cov. 55	0.241	**0.239**	0.391	-	0.246	0.243	1.460
	Cov. 60	0.248	0.245	0.391	-	0.261	**0.243**	1.396
Chr. 21	Cov. 25	**0.396**	**0.396**	0.536	0.513	**0.396**	**0.396**	1.943
	Cov. 30	**0.396**	**0.396**	0.466	-	**0.396**	**0.396**	1.728
	Cov. 35	**0.431**	**0.431**	0.489	0.500	0.442	**0.431**	1.723
	Cov. 40	**0.419**	**0.419**	0.511	0.500	**0.419**	0.430	1.694
	Cov. 45	**0.442**	**0.442**	0.534	-	**0.442**	0.465	1.599
	Cov. 50	**0.453**	**0.453**	0.546	0.569	**0.453**	0.465	1.363

Table 1: Switch error percentage on real Ashkenazim data, on chromosomes 1 and 21. The best result for each instance is boldfaced.

Simulated		HapCHAT		HapCol		WhatsHap		HapCUT2
Chr.	Avg. Cov.	Max25	Max30	Max25	Max30	Max15	Max20	
Chr. 1	Cov. 25	0.039	**0.035**	0.218	-	**0.035**	0.039	1.036
	Cov. 30	0.033	**0.028**	0.181	-	0.035	0.030	0.658
	Cov. 35	**0.028**	**0.028**	0.161	-	0.033	0.037	0.495
	Cov. 40	**0.024**	0.026	0.148	-	0.026	0.030	0.376
	Cov. 45	**0.022**	**0.022**	0.139	-	0.024	0.024	0.320
	Cov. 50	0.022	**0.020**	0.134	-	**0.020**	0.024	0.251
	Cov. 55	0.024	**0.022**	0.126	-	0.024	**0.022**	0.244
	Cov. 60	**0.017**	0.020	0.108	-	0.020	0.024	0.231
Chr. 21	Cov. 25	**0.035**	**0.035**	0.116	0.116	**0.035**	**0.035**	0.367
	Cov. 30	0.023	0.023	0.139	0.128	**0.012**	**0.012**	0.201
	Cov. 35	**0.023**	**0.023**	0.151	-	**0.023**	**0.023**	0.188
	Cov. 40	0.035	0.035	0.151	0.151	**0.023**	**0.023**	0.176
	Cov. 45	**0.058**	**0.058**	0.162	-	**0.058**	**0.058**	0.106
	Cov. 50	0.058	0.058	0.139	-	**0.035**	**0.035**	0.117

Table 2: Switch error percentage on simulated data, on chromosomes 1 and 21. The best result for each instance is boldfaced.

read the input (BAM) file, which is more-or-less the same for each method. Because we want to focus on comparing each method – preprocessing and phasing, and this reading step can be quite costly (Martin *et al.*, 2016), it only skews this analysis. The results are summarized in Tables 1–4 and Figure 1. The supplementary material contains more detailed results.

The quality of the predictions obtained with the experiments is summarized in Tables 1, 2 where the switch error percentage is reported. Each true haplotype is a mosaic of the predicted haplotypes. A switch error is the boundary (that is two consecutive SNP positions) between two portions of such a mosaic. The switch error percentage is the ratio between the minimum number of switch errors and the number of phased SNPs (expressed as a percentage). It is immediate to notice that both HapCol and HapCUT2 compute predictions that are not as good as those computed by HapCHAT or WhatsHap. Note that for HapCUT2, we only report the results of phasing without indels: the accuracy performance of phasing with indels was worse in all cases, while runtimes did not differ significantly. While the switch error percentage of HapCHAT and WhatsHap is close, we point out that HapCHAT computes the best prediction for almost all instances — the only exceptions are 3 instances on simulated of chromosome 21. On the other hand, HapCUT2 is the fastest tool, as reported in Tables 3, 4. Notice also that HapCol with maximum coverage 30 is unable to phase any instance on chromosome 1 and 5 out of 12 instances on chromosome 21 within the allotted 1-day timeout.

Since HapCHAT or WhatsHap produce the best predictions, we focus our analysis on those two tools. The results on chromosomes 1 and 21 are quite different. In fact, on chromosome 1 (both Ashkenazim and simulated datsets) HapCHAT always obtains the best phasings, which WhatsHap is able to match only in 5 (out of 14) instances. On the other hand, WhatsHap predictions are often close to those obtained by HapCHAT. On chromosome 21, WhatsHap and HapCHAT obtain the same results on 9 out of 12 instances. On the remaining 3 instances (all on simulated data) WhatsHap produces significantly better predictions than HapCHAT, but there is no clear cause for this behavior. The analysis of the running time shows that HapCHAT is much faster: at the largest maximum coverage for both tools, HapCHAT always run in less than half the time needed by WhatsHap. This suggests that, on chromosome 21, it is possible to run HapCHAT with maximum coverage 35 to obtain a new set of predictions. Finally, the memory usage of HapCHAT is always smaller than WhatsHap (when run with max coverage 20) and almost always smaller than HapCUT2. Moreover, HapCHAT has never required more than 4Mbytes of RAM, making it uninteresting to

Ashkenazim		HapCHAT		HapCol		WhatsHap		HapCUT2
Chr.	Avg. Cov.	Max25	Max30	Max25	Max30	Max15	Max20	
Chr. 1	Cov. 25	305	591	39456	-	418	8790	**46**
	Cov. 30	482	1292	46564	-	420	10178	**55**
	Cov. 35	635	2193	50071	-	435	11271	**64**
	Cov. 40	730	3095	50301	-	453	11775	**75**
	Cov. 45	833	3888	51570	-	455	11882	**82**
	Cov. 50	856	4579	53030	-	460	12079	**91**
	Cov. 55	925	5103	54012	-	487	12207	**105**
	Cov. 60	958	5550	53496	-	494	12355	**110**
Chr. 21	Cov. 25	82	185	7479	72947	84	1857	**13**
	Cov. 30	114	399	3323	-	83	2071	**15**
	Cov. 35	137	609	3701	32935	87	2146	**18**
	Cov. 40	158	766	3818	33337	89	2286	**20**
	Cov. 45	167	895	3914	-	92	2355	**23**
	Cov. 50	167	986	4268	32346	100	2376	**25**

Table 3: Time in seconds of the tools on Ashkenazim data, chromosomes 1 and 21. The best result for each instance is boldfaced.

deepen the analysis of RAM usage.

An analysis of Tables 1 and 2 towards finding the effect of average and maximum coverage shows that, on chromosome 1, increasing the maximum coverage results (unsurprisingly) in improved predictions. Considering larger datasets (that is larger average coverage) shows that there is a trend of improving predictions made by WhatsHap and HapCHAT with larger average coverage up to 50. For average coverages larger than 50, the predictions worsen. On chromosome 21, the maximum coverage has almost no effect (actually WhatsHap gives worse predictions for larger maximum coverages on 3 instances of Ashkenazim data), while the quality of the predictions worsen for larger datasets. This fact suggests that there is a tradeoff between the length of the chromosome and the number of reads to analyze that result in the best phasing.

The plots in Figure 1 represent the quality of the predictions computed by WhatsHap and HapCHAT as a function of the running time, for chromosome 1 on the Ashkenazim dataset. Besides the switch error rate, we have also investigated the Hamming distance, that is the number of calls that are different from the ground truth. Both plots confirm that HapCHAT computes predictions that are at least as good as those of WhatsHap (and clearly better in terms of Hamming distance) in a fraction of time.

Simulated		HapCHAT		HapCol		WhatsHap		HapCUT2
Chr.	Avg. Cov.	Max25	Max30	Max25	Max30	Max15	Max20	
Chr. 1	Cov. 25	313	572	38863	-	427	9306	**36**
	Cov. 30	479	1317	47367	-	427	10647	**42**
	Cov. 35	605	2167	18813	-	441	11132	**51**
	Cov. 40	697	3052	20007	-	456	12159	**58**
	Cov. 45	766	3754	56403	-	462	12023	**64**
	Cov. 50	803	4399	57135	-	462	12166	**72**
	Cov. 55	842	4882	56745	-	500	12397	**79**
	Cov. 60	835	5277	21070	-	485	12564	**86**
Chr. 21	Cov. 25	91	251	3321	29686	80	2008	**11**
	Cov. 30	117	456	3793	34198	83	2178	**12**
	Cov. 35	133	635	4535	-	88	2276	**15**
	Cov. 40	142	777	3848	32098	92	2240	**17**
	Cov. 45	148	894	4006	-	95	2383	**20**
	Cov. 50	150	937	4259	-	97	2349	**21**

Table 4: Time in seconds of the tools on simulated data, chromosomes 1 and 21. The best result for each instance is boldfaced.

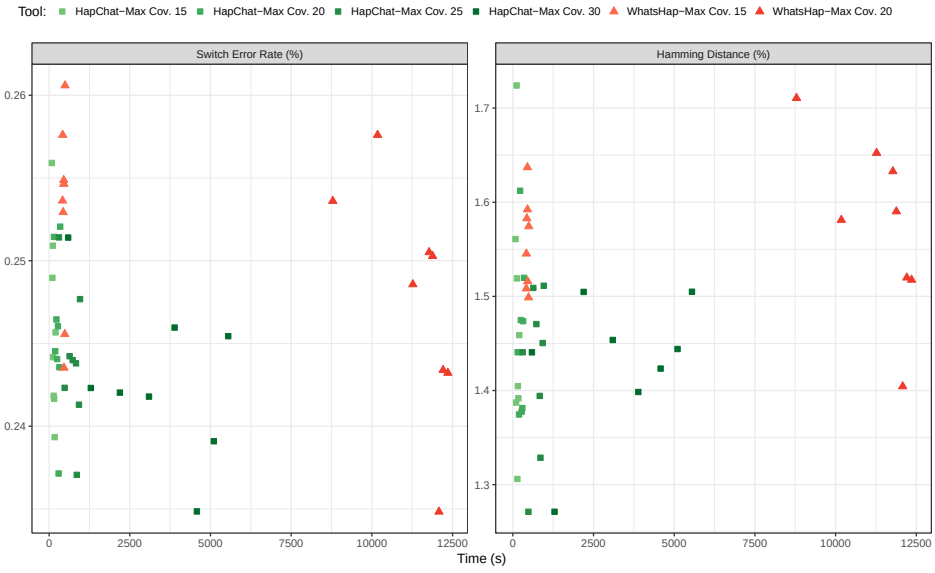

Figure 1: Switch error rate and Hamming distance as a function of running time, achieved by HapCHAT and WhatsHap at different max coverages on the Ashkenazim chromosome 1 dataset. For each tool and each max coverage we represent a point for each of the 8 possible values of the average coverage.

5 Conclusions

We have presented HapCHAT, a tool that is able to phase high coverage PacBio reads. We have compared HapCHAT with WhatsHap, HapCol, and HapCUT2 on real and simulated whole-chromosome datasets, with average coverage up to 60×. The real datasets have been taken from the GIAB project. Our experimental comparison shows that HapCHAT computes better predictions than the other tools, while requiring less time than all other tools, except for HapCUT2 (which is faster must gives worse predictions).

Adapting k allowed HapCHAT to achieve a better fit than HapCol of the number of corrections needed at each variant site. One could possibly obatin an even better fit by adapting k with backtracking: the need for correcting a number of errors at a given variant site could be met instead by correcting fewer errors upstream.

Acknowledgements

We would like to thank Tobias Marschall and Marcel Martin for several illuminating discussions.

Funding

We acknowledge the support of the Cariplo Foundation grant 2013–0955 (Modulation of anti cancer immune response by regulatory non-coding RNAs).

References

Amini, S., Pushkarev, D., Christiansen, L., *et al.* (2014). Haplotype-resolved whole genome sequencing by contiguity preserving transposition and combinatorial indexing. *Nature genetics*, **46**(12), 1343.

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

Bonizzoni, P., Della Vedova, G., Dondi, R., and Li, J. (2003). The haplotyping problem: An overview of computational models and solutions. *J. Comput. Sci. Technol.*, **18**(6), 675–688.

Bonizzoni, P., Dondi, R., Klau, G. W., *et al.* (2015). On the fixed parameter tractability and approximability of the minimum error correction problem. In *CPM*, volume 9133 of *LNCS*, pages 100–113.

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

Chen, Z.-Z., Deng, F., and Wang, L. (2013). Exact algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, **29**(16), 1938–45.

Delaneau, O. (2013). Haplotype estimation using sequencing reads. *American Journal of Human Genetics*, **93**, 687–696.

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

DePristo, M. A. *et al.* (2011). A framework for variation discovery and genotyping using next-generation dna sequencing data. *Nature Genetics*, **43**(5), 491–498.

Edge, P., Bafna, V., and Bansal, V. (2016). HapCUT2: robust and accurate haplotype assembly for diverse sequencing technologies. *Genome Research*, **27**(5), 801–812.

Fischer, Sarah O. and Marschall, Tobias (2016). Selecting Reads for Haplotype Assembly. *bioRxiv 046771*, doi:10.1101/046771

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

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

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

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

Ip, C., Loose, M., Tyson, J., *et al.* (2015). MinION analysis and reference consortium: Phase 1 data release and analysis. *F1000 Research*, **4**.

Kuleshov, V. (2014). Probabilistic single-individual haplotyping. *Bioinformatics*, **30**(17), i379–i385.

Kuleshov, V. *et al.* (2014). Whole-genome haplotyping using long reads and statistical methods. *Nature Biotechnology*, **32**(3), 261, 266.

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

Li, N. and Stephens, M. (2003). Modeling linkage disequilibrium and identifying recombination hotspots using single-nucleotide polymorphism data. *Genetics*, **165**(4), 2213–2233.

Lippert, R., Schwartz, R., Lancia, G., and Istrail, S. (2002). Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in Bioinformatics*, **3**(1), 23–31.

Loh, P., Danecek, P., Palamara, P., *et al.* (2016). Reference-based phasing using the haplotype reference consortium panel. *Nature Genetics*, **48**(11), 1443–1448.

Martin, M., Patterson, M., Garg, *et al.* (2016). WhatsHap: fast and accurate read-based phasing. *bioRxiv*, 085050.

Mazrouee, S. and Wang, W. (2014). FastHap: fast and accurate single individual haplotype reconstruction using fuzzy conflict graphs. *Bioinformatics*, **30**(17), 371–378.

O'Connell, J., Sharp, K., Shrine, N., *et al.* (2016). Haplotype estimation for biobank-scale data sets. *Nature Genetics*, **48**(7), 817–820.

Ono, Y. (2013). PBSIM: Pacbio reads simluator–toward accurate genome assembly. *Bioinformatics*, **29**, 119–121.

Patterson, M., Marschall, T., Pisanti, N., *et al.* (2014). WhatsHap: Haplotype assembly for future-generation sequencing reads. In *RECOMB*, volume 8394 of *LNCS*, 237–249.

Patterson, M., Marschall, T., Pisanti, N., *et al.* (2015). WhatsHap: Weighted haplotype assembly for future-generation sequencing reads. *J. of Computational Biology*, **6**(1), 498–509.

Pirola, Y., Zaccaria, S., Dondi, R., *et al.* (2015). HapCol: accurate and memory-efficient haplotype assembly from long reads. *Bioinformatics*. **32**(11), 1610–1617.

Roach, J., Glusman, G., Smit, A., *et al.* (2010). Analysis of genetic inheritance in a family quartet by whole-genome sequencing. *Science*, **328**(5978), 636–639.

Roberts, R. J., Carneiro, M. O., and Schatz, M. C. (2013). The advantages of SMRT sequencing. *Genome Biology*, **14**(6), 405.

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

Tewhey, R., Bansal, V., Torkamani, A., *et al.* (2011). The importance of phase information for human genomics. *Nature R. Genetics*, (3), 215–23.

van de Ven, M., Andressoo, J., van der Horst, G., *et al.* (2012). Effects of compound heterozygosity at the xpd locus on cancer and ageing in mouse models. *DNA Repair*, **11**(11), 874–83.

Zook, J. M. (2014). Integrating human sequence data sets provides a resource of benchmark snp and indel genotype calls. *Nature Biotechnology*, **32**, 246–251.