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

Stefano Beretta[1,†], Murray Patterson[1,†,*], Tobias Marschall[2,3], Marcel Martin[4], Simone Zaccaria[1,5], Gianluca Della Vedova[1], and Paola Bonizzoni[1,*]

[1] Department of Informatics, Systems, and Communication, University of Milano-Bicocca, Milano, Italy and
[2] Center for Bioinformatics, Saarland University, Saarbrücken, Germany and
[3] Department for Computational Biology and Applied Algorithmics, Max Planck Institute for Informatics, Saarbrücken, Germany and
[4] Science for Life Laboratory, Sweden and
[5] Department of Computer Science, Princeton University, Princeton, New Jersey, The United States of America

[†] 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

Background: Haplotype assembly is the process of assigning the different alleles of the Single Nucleotide Variants (SNVs) covered by mapped sequencing reads to the two haplotypes of the genome of a human individual. Long reads, which are nowadays cheaper to produce and more widely available than ever before, have been used to reduce the fragmentation of the assembled haplotypes since their ability to span several SNVs along the genome. These long reads are also characterized by a high error rate, an issue which may be mitigated, however, with larger sets of reads, when this error rate is uniform across genome positions. Unfortunately, current state-of-the-art methods that have been designed for long reads deal only with limited coverages.

Results: Here, we propose a new method for assembling haplotypes which combines and extends the features of previous approaches to deal with long reads and higher coverages. In particular, our algorithm is able to dynamically adapt the estimated number of errors at each variant site, while minimizing the total number of error corrections necessary for finding a

feasible solution. This allows our method to significantly reduce the required computational resources, allowing to consider datasets composed of higher coverages. The algorithm has been implemented in a freely available tool, HapCHAT: Haplotype Assembly Coverage Handling by Adapting Thresholds. An experimental analysis on sequencing reads with up to 60× coverage reveals accuracy improvements achieved by considering a higher coverage with lower runtimes.

Conclusions: Our method leverages the long-range information of sequencing reads that allows to obtain assembled haplotypes fragmented in a lower number of unphased haplotype blocks. At the same time, our method is also able to deal with higher coverages to better correct the errors in the original reads and to obtain more accurate haplotypes as a result.

Availability: HapCHAT is available at http://hapchat.algolab.eu under the GPL license.

Keywords: Single Individual Haplotyping, Long Reads, High Coverage, Haplotype Assembly, Minimum Error Correction

Contact: murray.patterson@unimib.it, bonizzoni@disco.unimib.it

Introduction

Due to the diploid nature of the human genome, *i.e.*, it has two copies of its genome, called *haplotypes*, genomic variants appear on either of these two copies. Knowing the specific haplotype on which each of the genomic variants 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).

More specifically, the variations between two haplotypes of the genome are, for the most part, in the form heterozygous Single Nucleotide Variants (SNVs), *i.e.*, single genome positions where the haplotypes contain two distinct alleles. Since a direct experimental reconstruction of the haplotypes is not yet cost effective Kuleshov et al. (2014a), computational methods aim to perform this task starting from sequencing reads mapped to a reference human genome. In fact, sequencing reads usually cover multiple SNV positions on the genome and, hence, they provide information about the corresponding

alleles that co-occur on a haplotype. In particular, *haplotype assembly* is the computational approach aiming to partition the reads into two sets such that all the reads belonging to the same set are assigned to the same haplotype.

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 computational methods for inferring the haplotypes of an individual are heuristics that statistically infer a phasing Li and Stephens (2003); Browning and Browning (2011) based on these panels. The accuracy of these methods, however, depends heavily on the size and diversity of the population used to compile the 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 the obvious solution, since it involves the assembly of the two haplotypes directly from the reads.

The combinatorial *Minimum Error Correction* (MEC) problem is the most commonly cited formulation of haplotype assembly Lippert et al. (2002). Under the principle of parsimony, MEC aims to find the minimum number of corrections to the values of sequencing reads in order to be able to unambiguously partition the reads into two haplotypes. Unfortunately, this problem is NP-hard Lippert et al. (2002) and it is even hard to approximate Cilibrasi et al. (2007); Bonizzoni et al. (2015, 2016). As such, several heuristics for haplotype assembly have been proposed during the time Panconesi and Sozio (2004); Bansal and Bafna (2008); Edge et al. (2016); Mazrouee and Wang (2014); Duitama et al. (2010), in particular HapCUT Bansal and Bafna (2008); Edge et al. (2016) has received particular attention from the community and is often quoted as the state-of-the-art method for haplotype assembly Kitzman et al. (2011, 2012); Selvaraj et al. (2013); Browning and Browning (2011). Several exact methods have also been proposed, including *Integer Linear Programming* (ILP) approaches Fouilhoux and Mahjoub (2012); Chen et al. (2013), and *Dynamic Programming* (DP) approaches that are *Fixed-Parameter Tractable* (FPT) in some parameter He et al. (2010); Bonizzoni et al. (2015). These methods achieve good results on datasets obtained using the traditional short sequencing reads. However, short reads do not allow to span more than a few SNV positions along the genome, rendering them inadequate for reconstructing long regions of the two haplotypes. In fact, the short range information provided by these reads does not allow to link many (if any) SNVs together. Consequently, the resulting assembled haplotypes are fragmented into many short haplotype blocks that remain unphased, relative to each other Chaisson et al. (2017).

The advent of third generation sequencing technologies introduces a

new kind of sequencing reads, called *long reads*, that are able to cover much longer portions of the genome Carneiro et al. (2012); Roberts et al. (2013). Each read may span several positions along the genome and the long-range information provided by these reads allow to link several SNVs. This results in the possibility of obtaining longer haplotype blocks that assign more variants to the corresponding haplotype Kuleshov et al. (2014b); Ip et al. (2015). Current third generation sequencing platforms offered by Pacific Biosciences (PacBio) Rhoads and Au (2015) and Oxford Nanopore Technologies (ONT) Jain et al. (2016) are now able to produce reads of up to 100 kilobasepairs (kbp) in length, and are much more capable of capturing together more variants than the short reads that are commonplace today. While PacBio technologies are characterized by a high error rate (substitution error rate up to $\sim$5% and indel rate up to $\sim$10%), this is uniformly distributed along the genome positions Carneiro et al. (2012); Roberts et al. (2013); Jain et al. (2015) – something we can take advantage of. Oxford Nanopore Technologies, on the other hand, have an even higher error rate which is also not uniformly distributed Cretu Stancu et al. (2017). Traditional approaches that have been designed for short reads fail when they are applied to these long reads, even when considering low coverages, as demonstrated in Patterson et al. (2014). This is due to the fact that these approaches scale poorly with increasing read length Chen et al. (2013); He et al. (2010).

Recently, two methods have been proposed to specifically deal with long reads and their characteristics, namely WhatsHap Patterson et al. (2014, 2015) and HapCol Pirola et al. (2016). On the one hand, WhatsHap introduces a dynamic programming algorithm that is fixed parameter tractable, with *coverage* as the parameter, where coverage is the maximum number of reads covering any genome position. Hence, this algorithm is able to leverage the long-range information of long reads since it is linear in the read length, but unfortunately it can deal only with datasets of limited coverages – up to 20$\times$, and hence resorts to pruning datasets with higher coverage Patterson et al. (2014). A parallel version of WhatsHap has been recently proposed showing the capability to deal with higher coverages $\sim$25$\times$ Bracciali et al. (2016). Although WhatsHap computes the theoretically optimal solution to the MEC problem, minimizing the overall number of corrections in the input reads, this could result, however, in columns having an unrealistically large number of corrections, which may not be coherent with how the errors are truly distributed in the actual reads.

On the other hand, HapCol proposes an approach that exploits the uniform distribution of sequencing errors characterizing long reads. In

particular, the authors propose a new formulation of the MEC problem where the maximum number of corrections is bounded in every column and is computed from the expected error rate Pirola et al. (2016). HapCol has been shown to be able to outperform WhatsHap, due to the capability to deal with datasets of higher coverages. However, the presence of genome positions containing more errors than expected (due to errors in the alignment or repetitive regions) is a problem for this approach. As a result, even HapCol was effectively limited to deal with instances of relatively small coverages up to $\sim$25–30$\times$, since even the presence of few outliers forces the algorithm to change the global behavior, or to fail.

As a result, both the methods proposed for haplotype assembly from long reads, WhatsHap and HapCol, have issues managing datasets with increasing coverages. However, considering a higher number of reads covering each position is indeed the most reliable way to face the high error rate characterizing the sequencing reads produced by third generation sequencing technologies. In fact, long reads share a limited number of errors on any given SNV position that they cover because errors are uniformly distributed across genome positions. Therefore, increasing the coverage mitigates the effects of sequencing errors and may allow to reconstruct haplotypes of higher quality.

In this work we propose a new method that combines and extends the main features of the previous WhatsHap and HapCol, and aims to deal with datasets of higher coverages while being robust to the presence of noise and outliers. In particular, we re-design the approach proposed in Pirola et al. (2016) by allowing also the dynamic adaption of the estimated error rate and, consequently, the maximum number of corrections that are allowed in each position. This allows the handling of columns that require more errors than expected, while avoiding exploring scenarios that involve a number of corrections that is much higher than necessary for a site. This is coupled with a merging procedure that merges pairs of reads that are highly likely to originate from the same haplotype, allowing this method to scale to significantly higher values of coverage. The method has been implemented in HapCHAT: Haplotype Assembly Coverage Handling by Adapting Thresholds that is freely available at http://github.com/AlgoLab/HapCHAT. An experimental analysis on real and simulated sequencing reads with up to 60$\times$ 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. (2016); the newest version of WhatsHap Martin et al. (2016), to which many features have since been

added; and HapCUT2 Bansal and Bafna (2008); Edge et al. (2016). We show that HapCHAT outperforms WhatsHap and HapCol in terms of required computational resources, while providing better accuracy predictions than all of the considered methods. These results confirm that high coverage is indeed necessary to deal with the high error rate of long reads in order to take advantage of their long-range information.

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, *i.e.*, 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 (SNVs). 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 Bonizzoni et al. (2015), 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. (2016). 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. (2016). 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. (2016); Bonizzoni et al. (2015).

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 in long read datasets. In fact, as reported in the original HapCol paper Pirola et al. (2016), the FPT algorithm is exponential in the number k of allowed corrections in each position. Therefore, we developed a preprocessing 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 reads (w.r.t. the original HapCol method), while reducing the runtimes.

Preprocessing

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 (average) probability that any single base has been read incorrectly (i.e. that a nucleotide in the input BAM file is wrong) — 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, where they agree on m of those sites and disagree on the other x sites. For this pair of reads, we compute a likelihood under the hypothesis that the reads originate from the same haplotype, and a likelihood under the hypothesis that the reads originate from different haplotypes. We then compute the ratio of these two likelihoods. This idea is similar to the one adopted in Kuleshov (2014), but our use is different.

Then, the probability of obtaining the two reads r_1 and r_2 under the hypothesis that they originate from the same haplotype is approximately $p_s(r_1, r_2) = (1-p)^{2m} p^x (1-p/3)^x$, that is we assume that we have no error in the shared part and exactly one error on the other sites. Similarly, the probability of obtaining the two reads r_1 and r_2 under the hypothesis that they originate from two different haplotypes is approximately $p_d(r_1, r_2) = p^m (1-p/3)^m (1-p/3)^x (1-p)^x$, that is we assume that there is exactly one error in the sites with same value and at most an error in the sites with different values.

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 want 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. This threshold was obtained empirically, in order to achieve the best performance in terms of quality of the predictions in the performed experimental analysis. 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. Notice that the merging heuristic of ProbHap Kuleshov (2014) considers only the ratio to determine when to merge two reads, while we analyze all pairs of reads to determine which *sets* of reads to merge.

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. (2016), 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). The idea is that the number of errors in a column j follows a binomial distribution, and hence we allow the smallest value k_j such that the probability of having more than k_j errors (with error rate ϵ) is at most α. This is done in order to bound the value of k, which is fundamental since HapCol implements an FPT algorithm that is exponential in the 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, when all solutions of an instance contain a column with more than k_j errors, HapCol is not able to find a solution. Therefore, we developed a heuristic procedure which has the final goal of guaranteeing that a solution is found, by slightly increasing the allowed number of errors beyond k_j, such that a solution exists for this number. 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 $z_{j,0} = k_j$ and $z_{j,i} = z_{j,i-1} + \lfloor \log_2(z_{j,i-1}) + 1 \rfloor$, *i.e.*, each term is obtained from the previous one by adding a logarithmic term, to guarantee that the number of allowed corrections does not grow too quickly. Then $k_j^* = \min_{i:D[j,C_{j,z_{j,i-1}}]\neq\infty} \{z_{j,i}\}$ 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 $z_{j,i}$ 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 $z_{j,i}$, 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.

Results and Discussion

We now describe the results of our experiments. In the first subsection, we describe the data that we use, or simulate. Then we detail the experiments that we set up in order to compare our tool with others in the next subsection. Finally, we present and discuss the results of these experiments.

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); Zook et al. (2016); Kalman et al. (2016). Since our goal is to assess the performance of different single-individual haplotype phasing methods, we study Chromosome 1 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 all the bi-allelic SNVs of the chromosome 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. For Chromosome 1, this resulted in 140744 SNVs, of which 48023 are heterozygous. We refer to this set of SNVs as the set of benchmark SNVs for the chromosome – the 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: Thanks to the availability of curated, high-quality reference panels, we use the output of a statistical phaser – a phasing of the benchmark SNVs – 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.

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 SNVs of the corresponding chromosome. It is the resulting phasing that we use as a benchmark for generating the simulated data, and comparing the different read-based phasing methods on this data.

For the real data, we used the latest high confidence trio-phased VCF of GIAB for hg37, available at ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/HG002_N
IllFB-IllGATKHC-Ion-10X-SOLID_CHROM1-22_v.3.3.2_highconf_triophased.vcf.gz
as the phasing benchmark.

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_
for Chromosome 1 of the son, which has an average coverage of 60.2× and an average mapped read lengths of 8687 bp. We then downsampled the read set to average coverages of 25×, 30×, 35×, 40×, 45×, 50×, 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 this set.

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 Chromosome 1. 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 the set of benchmark SNVs of the first haplotype phased by SHAPEIT, we incorporate them into the reference genome (hg19) by flipping the variant sites that are the alternative allele in this haplotype. The second haplotype is obtained analogously. Using these two true haplotypes

as the input, we produce a corresponding set of reads for this haplotype using PBSIM Hamada (2012), 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 Chromosome 1, we downsample to average coverages 25×, 30×, 35×, 40×, 45×, 50×, 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, for a total of 16 read sets, each in the form of a BAM file. It is on these 16 read sets, along with the the set of benchmark SNVs – a pair of VCF files – that we carry out our experiments, as described in the following section.

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. (2016), HapCUT2 Edge et al. (2016), ProbHap Kuleshov (2014), ReFHap Duitama et al. (2010) and FastHare Panconesi and Sozio (2004) by running them all on the data described in the previous subsection. Recall that, as detailed in the Introduction, that because WhatsHap, HapCol and HapCHAT are approaches with a core phasing algorithm that is FPT either in the coverage, or the number of errors at each SNV site, that coverage must first be reduced to some target maximum coverage before its core algorithm can be run. Each run of a tool on a dataset is given a time limit of one day, and a memory limit of 64GB. We now describe the details of how we parameterized each tool for comparison in what follows.

WhatsHap: For each of the 16 read sets, we provide to WhatsHap (version 0.13) the corresponding BAM and VCF file. We run WhatsHap on this input pair on otherwise default settings, with the exception of providing it the reference genome (hg19) via the optional parameter `--reference`. This allows WhatsHap to run in *realignment mode*, which has been shown to significantly boost accuracy predictions for noisy read sets such as PacBio,

as detailed in Martin et al. (2016). In particular, this mode is well suited to handle the abundant indel errors in the input reads. WhatsHap has a built-in read selection procedure Fischer and Marschall (2016) which subsequently prunes to a default maximum coverage of 15 before the core phasing algorithm is called. We then do another two parallel sets of runs of WhatsHap in realignment mode as above, but fixing to target maximum coverages 20 and 25, respectively, by providing the additional optional parameter `-H 20` and `-H 25`, respectively. We did not further attempt any higher maximum coverages, because at maximum coverage 25, WhatsHap either exceeded one day of runtime or 64GB of memory on each of the 16 datasets. It is the resulting set of 48 phasings by WhatsHap, in the form of phased VCF, that we use for the basis of comparison with the other methods.

HapCol: For each of the 16 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.* the Introduction) – 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 64 input files, we run HapCol with its default value of $\alpha = 0.01$ (and of $\varepsilon = 0.05$) (*cf.* the subsection on Adaptive k-cMEC or or Pirola et al. (2016) for details on the meaning of α and ε). Since HapCol is not adaptive, but we want to give it a chance to obtain a solution on its instance, should a given α be infeasible (*cf.* the subsection on Adaptive k-cMEC), 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. Note that we did not further attempt any higher maximum coverages, because at maximum coverage 30, HapCol either exceeded one day of runtime or 64GB of memory on each of the 16 datasets. It is this set of resulting 64 phasings (phased VCF files) that we use to compare with the other methods.

ProbHap, RefHap and FastHare: For each of the 16 read sets, we use the `extractHAIRS` program that is distributed with the original HapCut Bansal and Bafna (2008) to convert its BAM / VCF pair into the custom input format for these methods. We also extract another 16 sets of parallel inputs as above, but also providing the additional optional parameter `--indels 1`, which also extracts reads that span indels. We then ran each method on these 32 instances with default settings, each producing a

custom input which is then converted to a phased VCF with the subcommand `hapcut2vcf` of the WhatsHap toolbox. We report, for each tool, the results for the option (`--indels` or not) which performed the best in terms of prediction switch error.

HapCUT2: For each of the 16 read sets, we use the `extractHAIRS` program that comes with HapCUT2 to convert its BAM / VCF pair into the custom input format for HapCUT2. We also extract another 16 sets of parallel inputs for HapCUT2 as above, but also providing the additional optional parameter `--indels 1`, which also extracts reads that span indels. For both of these parameter combinations, we also ran `extractHAIRS` with parameter `--pacbio 1`, which activates a newly-developed realignment procedure for pacbio reads. Because of a crash of this `extractHAIRS` on the real data, we could only provide results on the simulated data for HapCUT2. We then ran HapCUT2 on these 16 simulated instances with default settings each producing a custom output which is then converted to phased VCF with the subcommand `hapcut2vcf` of the WhatsHap toolbox. We provide a Supplementary Material, where we ran HapCUT2 on the output of the stable `extractHAIRS` that is distributed with the original HapCut (but without realignment), to allow it to complete on the real instances as well.

HapCHAT: For each of the 16 read sets, we provide to HapCHAT the corresponding BAM and VCF file. We run HapCHAT on this input pair on otherwise default settings, with the exception of providing it the reference genome (hg19) via the optional parameter `--reference`. This allows HapCHAT to run in realignment mode like with WhatsHap, thanks to partial integration of HapCHAT into the WhatsHap codebase. We then apply our merging step as described in the subsection Preprocessing, 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×. It is the resulting 64 phasings, in phased VCF format, for which the comparison of HapCHAT to other methods is based.

Experimental results and discussion

The times reported here do not include the time necessary to read the input (BAM) file, which is more-or-less the same for each method. The results are summarized in Tables 1–4 and Figure 1.

The quality of the predictions obtained with the experiments is summarized in Tables 1, 2 where the switch error percentage and the Hamming distance percentage are reported. Each true haplotype is a mosaic of the predicted haplotypes. A switch error is the boundary (that is two consecutive SNV positions) between two portions of such a mosaic. The switch error percentage is the ratio between the number of switch errors and the number of phased SNPs minus one (expressed as a percentage). It is immediate to notice that HapCHAT, WhatsHap, and HapCUT2 (on the simulated dataset) compute the best predictions. Notice that HapCUT2 requires the new option `--indels` (which includes a realignment step) to obtain predictions that are roughly comparable to those of HapCHAT and WhatsHap. Still, this option makes HapCUT2 crash on the Ashkenazim dataset. 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. On the other hand, HapCUT2 is the fastest tool, as reported in Tables 3, 4.

Although the switch error is one of the most adopted measures used to evaluate the quality of the phased haplotypes, we decided to also employ another measure which could better take into account the correctly phased positions of the haplotypes. More precisely, for each haplotype block, we computed the Hamming distance percentage on the predictions of the tested datasets, which corresponds to the ratio between the Hamming distances of the phased haplotypes and the number of phased SNVs (expressed as a percentage). Here we report the average of these percentages, over the set of haplotype blocks. Observing the results reported in Table 1 and Table 2, HapCHAT always (except in one case) achieves the best results in terms of Hamming distance percentage. This highlights the fact that each haplotype reconstructed by HapCHAT is closer to the real one, with respect to the individual SNV positions. We have to point out the Hamming distance varies more wildly than the Switch error among all distances. At the same time, HapCHAT is the tools that has the most consistent behaviour in that regard.

Since HapCHAT or WhatsHap produce the best predictions, we focus our analysis on those two tools. On the tested Chromosome 1 (both Ashkenazim and simulated datasets), HapCHAT almost always obtains the best phasings, which WhatsHap is able to match only in 5 (out of 24) instances. On the other hand, WhatsHap predictions are often close to those obtained by HapCHAT.

The analysis of the running time shows that HapCHAT is much faster:

at the largest maximum coverage for both tools, HapCHAT always runs in less than half the time needed by WhatsHap. 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 4GB of RAM, making it uninteresting to deepen the analysis of RAM usage. An overview on the comparison of running time is given in Figure 1. As it is possible to observe from Tables 3 and 4, that although both time and memory used by HapCHAT is growing with the (average) coverage, the rate at which the time increments with higher coverage is decreasing. Similarly, also the memory increment is almost linear with respect to the growth of the coverage of the datasets. On the other hand, while the changes of time and memory required by HapCol and WhatsHap to process higher coverages remain similar, they are always higher than HapCHAT at any value of average coverage. Contrary to HapCHAT, because HapCol and WhatsHap are not adaptive (see intro for more details), they must be run at a uniform *maximum* coverage of 25 and 20, respectively, and exhibit similar runtimes and memory usage for all datasets, because they are not responding to the increasing average coverage. HapCHAT, on the other hand, processes these datasets at the higher uniform maximum coverage of 30, and because it *adapts* to this increased average coverage, we see this linear trend in increased resource usage, as expected. Finally, we point out that HapCUT2, ReFHap, and FastHare require always the same memory, since it does not depend on the coverage, and the time grows linearly, while ProbHap exhibits a behavior reflecting the coverage increment, especially in terms of memory consumption, but the values are not comparable to that of HapCHAT.

An analysis of Tables 1 and 2 towards finding the effect of average and maximum coverage shows that 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.

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 the time.

Finally, Table 5 reports for each of the 16 datasets, the SNV sites

when the adaptive procedure of subsection Adaptive k-cMEC was activated. Interestingly, only the Simulated dataset needed corrections at the more elevated level of '5 to 8', indicating that it contains more unanticipated errors than the real datasets. Indeed this demonstrates that this adaptive procedure is an improvement over HapCol, recalling that each time this procedure is invoked, HapCol fails by definition. An added benefit of this procedure is that it can serve as an indicator as to the quality of the read set to be phased.

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 but gives much worse predictions).

Introducing the capability of adapting the number of errors permitted in each column allows HapCHAT to achieve a better fit than HapCol of the number of corrections needed at each variant site. Still, the current approach allows such adaptation only for the current column. Coupling this step with backtracking could result in fewer overall corrections.

Another direction of research is to fully consider the parent-sibling relations in trios. This is especially relevant, since most of the GIAB data is on trios.

Finally, we are working on the integration of HapCHAT with the WhatsHap tool to provide a more powerful haplotype phasing method able to combine the strengths of the two approaches.

Declarations

Ethics approval and consent to participate: Not applicable

Consent for publication: Not applicable

Availability of data and materials: The PacBio long reads data that we used are publicly available at ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/AshkenazimTrio/HG002_NA

The simulated datasets that we have used can be downloaded at: `https://drive.google.com/drive/folders/0BxqLPsY2hmAXMlowZF9JQllZNEU`. The BAM files are in archive`HapCHAT-experiments_bam--simulated.tar.gz`, while the corresponding VCFs can be found at `HapCHAT-experiments.tar.gz`.

Instructions on how to use and replicate the experiments can be found at `http://hapchat.algolab.eu`

Competing interests: The authors declare that they have no competing interests.

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

Acknowledgments: Not applicable

References

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

Paola Bonizzoni, Gianluca Della Vedova, Riccardo Dondi, and Jing Li. The haplotyping problem: An overview of computational models and solutions. *Journal of Computer Science and Technolgy*, 18(6):675–688, 2003.

Paola Bonizzoni, Riccardo Dondi, Gunnar W Klau, Yuri Pirola, Nadia Pisanti, and Simone Zaccaria. On the fixed parameter tractability and approximability of the minimum error correction problem. In *26th Annual Symposium on Combinatorial Pattern Matching (CPM)*, volume 9133 of *LNCS*, pages 100–113, 2015.

Paola Bonizzoni, Riccardo Dondi, Gunnar W Klau, Yuri Pirola, Nadia Pisanti, and Simone Zaccaria. On the minimum error correction problem for haplotype assembly in diploid and polyploid genomes. *Journal of Computational Biology*, 23(9):718–736, 2016.

Andrea Bracciali, Marco Aldinucci, Murray Patterson, Tobias Marschall, Nadia Pisanti, Ivan Merelli, and Massimo Torquati. PWHATSHAP: efficient haplotyping for future generation sequencing. *BMC Bioinformatics*, 17(11):342, 2016. doi: 10.1186/s12859-016-1170-y.

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

Mauricio O. Carneiro, Carsten Russ, Michael G. Ross, Stacey B. Gabriel, Chad Nusbaum, and Mark A. DePristo. Pacific Biosciences sequencing technology for genotyping and variation discovery in human data. *BMC Genomics*, 13(1):375, 2012.

Mark J.P. Chaisson, Ashley D. Sanders, Xuefang Zhao, Ankit Malhotra, David Porubsky, Tobias Rausch, Eugene J. Gardner, Oscar Rodriguez, Li Guo, Ryan L. Collins, Xian Fan, Jia Wen, Robert E. Handsaker, Susan Fairley, Zev N. Kronenberg, Xiangmeng Kong, Fereydoun Hormozdiari, Dillon Lee, Aaron M. Wenger, Alex Hastie, Danny Antaki, Peter Audano, Harrison Brand, Stuart Cantsilieris, Han Cao, Eliza Cerveira, Chong Chen, Xintong Chen, Chen-Shan Chin, Zechen Chong, Nelson T. Chuang, Deanna M. Church, Laura Clarke, Andrew Farrell, Joey Flores, Timur Galeev, Gorkin David, Madhusudan Gujral, Victor Guryev, William Haynes-Heaton, Jonas Korlach, Sushant Kumar, Jee Young Kwon, Jong Eun Lee, Joyce Lee, Wan-Ping Lee, Sau Peng Lee, Patrick Marks, Karine Valud-Martinez, Sascha Meiers, Katherine M. Munson, Fabio Navarro, Bradley J. Nelson, Conor Nodzak, Amina Noor, Sofia Kyriazopoulou-Panagiotopoulou, Andy Pang, Yunjiang Qiu, Gabriel Rosanio, Mallory Ryan, Adrian Stutz, Diana C.J. Spierings, Alistair Ward, AnneMarie E. Welsch, Ming Xiao, Wei Xu, Chengsheng Zhang, Qihui Zhu, Xiangqun Zheng-Bradley, Goo Jun, Li Ding, Chong Lek Koh, Bing Ren, Paul Flicek, Ken Chen, Mark B. Gerstein, Pui-Yan Kwok, Peter M. Lansdorp, Gabor Marth, Jonathan Sebat, Xinghua Shi, Ali Bashir, Kai Ye, Scott E. Devine, Michael Talkowski, Ryan E. Mills, Tobias Marschall, Jan Korbel, Evan E. Eichler, and Charles Lee. Multi-platform discovery of haplotype-resolved structural variation in human genomes. *bioRxiv*, 2017. doi: 10.1101/193144.

Zhi-Zhong Chen, Fei Deng, and Lusheng Wang. Exact algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 29(16): 1938–45, 2013.

Rudi Cilibrasi, Leo Van Iersel, Steven Kelk, and John Tromp. The complexity of the single individual SNP haplotyping problem. *Algorithmica*, 49(1): 13–36, 2007.

Mircea Cretu Stancu, Markus J. van Roosmalen, Ivo Renkens, Marleen M. Nieboer, Sjors Middelkamp, Joep de Ligt, Giulia Pregno, Daniela Gi-

achino, Giorgia Mandrile, Jose Espejo Valle-Inclan, Jerome Korzelius, Ewart de Bruijn, Edwin Cuppen, Michael E. Talkowski, Tobias Marschall, Jeroen de Ridder, and Wigard P. Kloosterman. Mapping and phasing of structural variation in patient genomes using nanopore sequencing. *Nature Communications*, 8(1326), 2017. doi: 10.1038/s41467-017-01343-4.

Olivier Delaneau. Haplotype estimation using sequencing reads. *American Journal of Human Genetics*, 93:687–696, 2013.

Jorge Duitama, Thomas Huebsch, Gayle McEwen, Eun-Kyung Suk, and Margret R Hoehe. ReFHap: a reliable and fast algorithm for single individual haplotyping. In *BCB*, pages 160–169. ACM, 2010.

Peter Edge, Vineet Bafna, and Vikas Bansal. HapCUT2: robust and accurate haplotype assembly for diverse sequencing technologies. *Genome Research*, 213462(116), 2016. doi: 10.1101/gr.213462.116.

Sarah O. Fischer and Tobias Marschall. Selecting Reads for Haplotype Assembly. *bioRxiv*, page 046771, 2016. doi: 10.1101/046771. URL `http://www.biorxiv.org/content/early/2016/04/06/046771`.

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

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

Gustavo Glusman, Hannah C. Cox, and Jared C. Roach. Whole-genome haplotyping approaches and genomic medicine. *Genome Medicine*, 6(9): 73, 2014. doi: 10.1186/s13073-014-0073-7.

Yukiteru Ono Kiyoshi Asai Michiaki Hamada. PBSIM: Pacbio reads simluator–toward accurate genome assembly. *Bioinformatics*, 29:119–121, 2012.

Dan He, Arthur Choi, Knot Pipatsrisawat, Adnan Darwiche, and Eleazar Eskin. Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, 26(12):i183–i190, 2010.

Camilla L.C. Ip, Matthew Loose, John R. Tyson, Mariateresa de Cesare, Bonnie L. Brown, Miten Jain, Richard M. Leggett, David A. Eccles, Vadim Zalunin, John M. Urban, Paolo Piazza, Rory J. Bowden, Benedict

Paten, Solomon Mwaigwisya, Elizabeth M. Batty, Jared T. Simpson, Terrance P. Snutch, Ewan Birney, David Buck, Sara Goodwin, Hans J. Jansen, Justin O'Grady, and Hugh E. Olsen. MinION analysis and reference consortium: Phase 1 data release and analysis. *F1000 Research*, 4, 2015. doi: 10.12688/f1000research.7201.1.

Miten Jain, Ian T. Fiddes, Karen H. Miga, Hugh E. Olsen, Benedict Paten, and Mark Akeson. Improved data analysis for the minion nanopore sequencer. *Nature methods*, 12:351–356, 2015.

Miten Jain, Hugh E. Olsen, Benedict Paten, and Mark Akeson. The oxford nanopore minion: delivery of nanopore sequencing to the genomics community. *Genome Biology*, 17(1):239, 2016.

Lisa Kalman, Vivekanada Datta, Mickey Williams, Justin M. Zook, Marc L. Salit, and Jin-Yeong Han. Development and characterization of reference materials for genetic testing: Focus on public partnerships. *Annals of Laboratory Medicine*, 36(6):513–520, 2016. doi: 10.3343/alm.2016.36.6.513.

Jacob O. Kitzman, Alexandra P. MacKenzie, Andrew Adey, Joseph B. Hiatt, Rupali P. Patwardhan, Peter H. Sudmant, Sarah B. Ng, Can Alkan, Ruolan Qiu, Evan E. Eichler, et al. Haplotype-resolved genome sequencing of a gujarati indian individual. *Nature Biotechnology*, 29(1):59–63, 2011.

Jacob O. Kitzman, Matthew W. Snyder, Mario Ventura, Alexandra P. Lewis, Ruolan Qiu, LaVone E. Simmons, Hilary S. Gammill, Craig E. Rubens, Donna A. Santillan, Jeffrey C. Murray, et al. Noninvasive whole-genome sequencing of a human fetus. *Science Translational Medicine*, 4(137): 137ra76–137ra76, 2012.

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

Volodymyr Kuleshov, Dan Xie, Rui Chen, Dmitry Pushkarev, Zhihai Ma, Tim Blauwkamp, Michael Kertesz, and Michael Snyder. Whole-genome haplotyping using long reads and statistical methods. *Nature Biotechnology*, 32(3):261–266, 2014a.

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

Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *ArXiv e-prints*, March 2013.

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

Ross Lippert, Russell Schwartz, Giuseppe Lancia, and Sorin Istrail. Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Briefings in Bioinformatics*, 3(1):23–31, 2002.

Po-Ru Loh, Petr Danecek, Pier Francesco Palamara, Christian Fuchsberger, Yakir A. Reshef, Hilary K. Finucane, Sebastian Schoenherr, Lukas Forer, Shane McCarthy, Goncalo R. Abecasis, Richard Durbin, and Alkes L. Price. Reference-based phasing using the haplotype reference consortium panel. *Nature Genetics*, 48(11):1443–1448, 2016. doi: 10.1038/ng.3679.

Marcel Martin, Murray Patterson, Shilpa Garg, Sarah O. Fischer, Nadia Pisanti, Gunnar W. Klau, Alexander Schoenhuth, and Tobias Marschall. WhatsHap: fast and accurate read-based phasing. 2016.

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

Jared O'Connell, Kevin Sharp, Nick Shrine, Louise Wain, Ian Hall, Martin Tobin, Jean-Francois Zagury, Olivier Delaneau, and Jonathan Marchini. Haplotype estimation for biobank-scale data sets. *Nature Genetics*, 48(7): 817–820, 2016. doi: 10.1038/ng.3583.

Alessandro Panconesi and Mauro Sozio. Fast hare: A fast heuristic for single individual SNP haplotype reconstruction. In *Algorithms in Bioinformatics, 4th International Workshop, WABI 2004, Bergen, Norway, September 17-21, 2004, Proceedings*, pages 266–277, 2004.

Murray Patterson, Tobias Marschall, Nadia Pisanti, Leo van Iersel, Leen Stougie, Gunnar W. Klau, and Alexander Schönhuth. WhatsHap: Haplotype assembly for future-generation sequencing reads. In *RECOMB*, volume 8394 of *LNCS*, pages 237–249, 2014.

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. *Journal of Computational Biology*, 6(1):498–509, 2015.

Yuri Pirola, Simone Zaccaria, Riccardo Dondi, Gunnar W. Klau, Nadia Pisanti, and Paola Bonizzoni. HapCol: accurate and memory-efficient haplotype assembly from long reads. *Bioinformatics*, 32(11):1610–1617, 2016. doi: 10.1093/bioinformatics/btv495.

Anthony Rhoads and Kin Fai Au. Pacbio sequencing and its applications. *Genomics, Proteomics and Bioinformatics*, 13(5):278–289, 2015.

Jared C. Roach, Gustavo Glusman, Arian F.A. Smit, Chad D. Huff, Robert Hubley, Paul T. Shannon, Lee Rowen, Krishna P. Pant, Nathan Goodman, Michael Bamshad, Jay Shendure, Radoje Drmanac, Lynn B. Jorde, Leroy Hood, and David J. Galas. Analysis of genetic inheritance in a family quartet by whole-genome sequencing. *Science*, 328(5978):636–639, 2010. doi: 10.1126/science.1186802.

Richard J. Roberts, Mauricio O. Carneiro, and Michael C. Schatz. The advantages of SMRT sequencing. *Genome Biology*, 14(6):405, 2013.

Siddarth Selvaraj, Jesse R. Dixon, Vikas Bansal, and Bing Ren. Whole-genome haplotype reconstruction using proximity-ligation and shotgun sequencing. *Nature Biotechnology*, 31(12):1111–1118, 2013.

Ryan Tewhey, Vikas Bansal, Ali Torkamani, Eric J. Topol, and Nicholas J. Schork. The importance of phase information for human genomics. *Nature Reviews Genetics*, (3):215–223, 2011. doi: 10.1038/nrg2950.

Justin M. Zook. Integrating human sequence data sets provides a resource of benchmark SNP and indel genotype calls. *Nature Biotechnology*, 32: 246–251, 2014.

Justin M. Zook, David Catoe, Jennifer McDaniel, Lindsay Vang, Noah Spies, Arend Sidow, Ziming Weng, Yuling Liu, Christopher E. Mason, Noah Alexander, Elizabeth Henaff, Alexa B.R. McIntyre, Dhruva Chandramohan, Feng Chen, Erich Jaeger, Ali Moshrefi, Khoa Pham, William Stedman, Tiffany Liang, Michael Saghbini, Zeljko Dzakula, Alex Hastie, Han Cao, Gintaras Deikus, Eric Schadt, Robert Sebra, Ali Bashir, Rebecca M. Truty, Christopher C. Chang, Natali Gulbahce, Keyan Zhao, Srinka Ghosh, Fiona Hyland, Yutao Fu, Mark Chaisson, Chunlin Xiao, Jonathan Trow, Stephen T. Sherry, Alexander W. Zaranek, Madeleine Ball, Jason Bobe, Preston Estep, George M. Church, Patrick Marks, Sofia Kyriazopoulou-Panagiotopoulou, Grace X.Y. Zheng, Michael Schnall-Levin, Heather S. Ordonez, Patrice A. Mudivarti, Kristina Giorda, Ying Sheng,

Karoline Bjarnesdatter Rypdal, and Marc Salit. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Scientific Data*, 3(160025), 2016. doi: 10.1038/sdata.2016.25.

Ashkenazim Chr.1 Measure	Avg. Cov.	HapCHAT	HapCol	WhatsHap	HapCUT2	ReFHap	ProbHap	FastHare
Switch Error (%)	Cov. 25	**0.334**	0.662	0.342	-	2.813	3.303	3.547
	Cov. 30	**0.324**	0.623	0.333	-	2.420	2.980	3.133
	Cov. 35	**0.320**	0.601	0.332	-	2.221	2.831	2.933
	Cov. 40	**0.324**	0.575	0.332	-	2.027	3.053	2.691
	Cov. 45	**0.323**	0.533	0.336	-	1.932	2.885	2.522
	Cov. 50	**0.323**	0.490	**0.323**	-	1.864	2.839	2.303
	Cov. 55	**0.323**	0.452	0.331	-	1.774	2.669	2.268
	Cov. 60	0.327	0.452	**0.322**	-	1.740	2.847	2.123
Hamming Distance (%)	Cov. 25	**0.54**	2.41	0.84	-	3.96	3.42	5.53
	Cov. 30	**0.35**	2.18	0.60	-	3.46	3.41	5.38
	Cov. 35	**0.36**	2.02	0.42	-	3.99	3.89	5.62
	Cov. 40	**0.37**	1.66	0.44	-	3.10	3.69	5.08
	Cov. 45	**0.38**	1.80	0.42	-	3.02	3.00	4.49
	Cov. 50	0.41	1.47	**0.38**	-	2.84	2.81	4.32
	Cov. 55	**0.40**	0.87	0.41	-	3.28	3.21	4.67
	Cov. 60	0.39	1.25	**0.36**	-	3.60	3.60	5.06

Table 1: Switch error percentage and Hamming Distance on real Ashkenazim dataset of Chromosome 1. For each dataset, its row identified by its average coverage (Avg. Cov.), we report the results obtained by running the tools with the maximum coverage we could test: 30× for HapCHAT, 25× for HapCol, and 20× for WhatsHap, according the time and memory limits imposed – no bound was set for HapCUT2, ReFHap, ProbHap, and FastHare. The best result for each dataset is boldfaced.

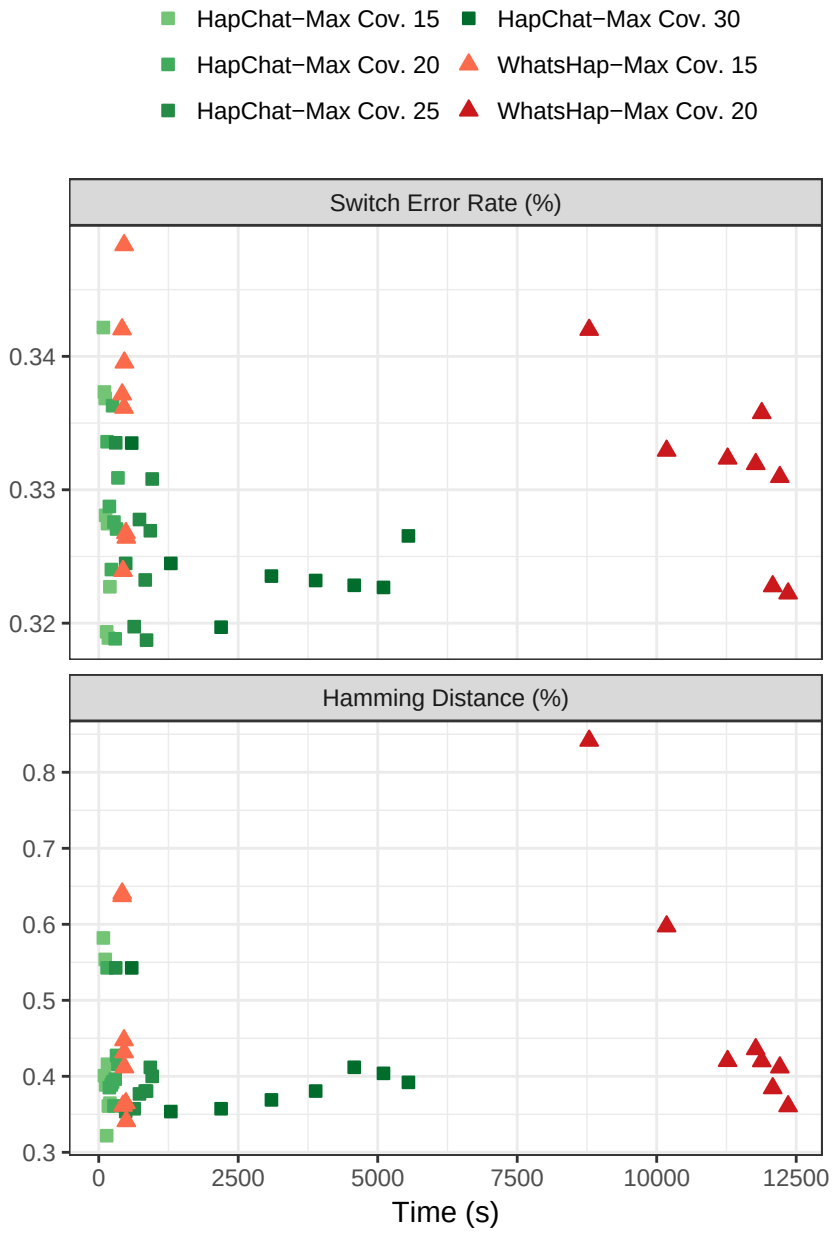

Figure 1: **Switch error rate and Hamming distance as a function of running time**. As achieved by HapCHAT and WhatsHap at different maximum coverages on the real Ashkenazim Chromosome 1 dataset. For each tool and each maximum coverage, we represent a point for each of the 8 possible values of the average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

Simulated Chr.1 Measure.	Avg. Cov.	HapCHAT	HapCol	WhatsHap	HapCUT2	ReFHap	ProbHap	FastHare
Switch Error (%)	Cov. 25	**0.035**	0.218	0.039	0.037	1.081	1.487	2.112
	Cov. 30	**0.028**	0.181	0.030	0.037	0.725	1.166	1.430
	Cov. 35	**0.028**	0.161	0.037	0.037	0.537	0.879	1.086
	Cov. 40	**0.026**	0.148	0.030	0.037	0.425	0.698	0.901
	Cov. 45	**0.022**	0.139	0.024	**0.022**	0.404	0.834	0.781
	Cov. 50	0.020	0.134	0.024	**0.018**	0.324	0.800	0.586
	Cov. 55	0.022	0.126	0.022	**0.018**	0.273	0.733	0.565
	Cov. 60	**0.020**	0.108	0.024	0.022	0.248	0.742	0.470
Hamming Distance (%)	Cov. 25	0.35	1.30	0.42	**0.33**	3.43	2.66	5.95
	Cov. 30	**0.33**	1.92	0.42	0.38	2.52	2.43	4.55
	Cov. 35	**0.27**	1.37	0.55	0.37	1.92	2.09	3.93
	Cov. 40	**0.26**	1.18	0.41	0.27	1.86	1.67	3.31
	Cov. 45	0.34	1.02	0.34	**0.27**	1.95	2.07	3.12
	Cov. 50	**0.27**	1.18	0.81	0.73	1.42	1.93	3.14
	Cov. 55	0.28	1.13	0.76	**0.20**	1.48	1.72	3.19
	Cov. 60	**0.13**	1.26	0.16	0.57	1.49	1.68	3.49

Table 2: Switch error percentage and Hamming Distance percentage on simulated datasets of Chromosome 1. For each dataset, its row identified by its average coverage (Avg. Cov.), we report the results obtained by running the tools with the maximum coverage we could test: 30× for HapCHAT, 25× for HapCol, and 20× for WhatsHap, according the time and memory limits imposed – no bound was set for HapCUT2, ReFHap, ProbHap, and FastHare. The best result for each dataset is boldfaced.

Ashkenazim Chr.1 Measure	Avg. Cov.	HapCHAT	HapCol	WhatsHap	HapCUT2	ReFHap	ProbHap	FastHare
Time (sec.)	Cov. 25	591	39456	8790	-	80	43573	3
	Cov. 30	1292	46564	10178	-	196	72696	4
	Cov. 35	2193	50071	11271	-	308	86400	4
	Cov. 40	3095	50301	11775	-	499	86400	5
	Cov. 45	3888	51570	11882	-	822	86400	6
	Cov. 50	4579	53030	12079	-	1192	86400	8
	Cov. 55	5103	54012	12207	-	1777	86400	9
	Cov. 60	5550	53496	12355	-	2493	86400	13
Memory (MB)	Cov. 25	1370	2263	5511	-	3005	4693	3005
	Cov. 30	1661	2562	6195	-	3005	5355	3005
	Cov. 35	1966	2908	6514	-	3005	6312	3005
	Cov. 40	2291	3231	6483	-	3005	5548	3005
	Cov. 45	2636	3190	6937	-	3005	6064	3005
	Cov. 50	3158	3286	7144	-	3005	6399	3005
	Cov. 55	3549	3479	7229	-	3005	6719	3005
	Cov. 60	3968	5412	7430	-	3005	6323	3005

Table 3: Time in seconds and peak of RAM usage in Megabytes of the tools on real Ashkenazim datasets of Chromosome 1. For each dataset, its row identified by its average coverage (Avg. Cov.), we report the results obtained by running the tools with the maximum coverage we could test: 30× for HapCHAT, 25× for HapCol, and 20× for WhatsHap, according the time and memory limits imposed – no bound was set for HapCUT2, ReFHap, ProbHap, and FastHare.

Simulated Chr.1 Measure	Avg. Cov.	HapCHAT	HapCol	WhatsHap	HapCUT2	ReFHap	ProbHap	FastHare
Time (sec.)	Cov. 25	572	38863	9306	205	44	33988	3
	Cov. 30	1317	47367	10647	238	91	56165	3
	Cov. 35	2167	18813	11132	286	167	80061	4
	Cov. 40	3052	20007	12159	323	269	86400	5
	Cov. 45	3754	56403	12023	367	423	86400	6
	Cov. 50	4399	57135	12166	412	672	86400	6
	Cov. 55	4882	56745	12397	467	1019	86400	7
	Cov. 60	5277	21070	12564	496	1536	86400	9
Memory (MB)	Cov. 25	1378	2180	5162	3262	3007	4284	3007
	Cov. 30	1667	4187	6118	3266	3008	5320	3008
	Cov. 35	1984	2134	6559	3270	3008	5709	3008
	Cov. 40	2315	2186	6780	3272	3009	6366	3009
	Cov. 45	2665	5037	7044	3276	3010	6532	3010
	Cov. 50	3180	5223	7058	3279	3010	6432	3010
	Cov. 55	3591	5483	7213	3282	3011	6657	3011
	Cov. 60	4009	2374	7295	3286	3011	6606	3011

Table 4: Time in seconds and peak of RAM usage in Megabytes of the tools on simulated datasets of Chromosome 1. For each dataset, its row identified by its average coverage (Avg. Cov.), we report the results obtained by running the tools with the maximum coverage we could test: 30× for HapCHAT, 25× for HapCol, and 20× for WhatsHap, according the time and memory limits imposed – no bound was set for HapCUT2, ReFHap, ProbHap, and FastHare.

Chr.1 Data	Avg. Cov.	4 to 7	5 to 8
Ashkenazim	Cov. 25		
	Cov. 30		
	Cov. 35	35556	
	Cov. 40		
	Cov. 45	35581	
	Cov. 50	35593, 42897	
	Cov. 55	3528	
	Cov. 60	46338, 46339	
Simulated	Cov. 25	35569, 38788	26778
	Cov. 30	35594, 38815, 38817	26800
	Cov. 35	38827	26811
	Cov. 40	38837	38834, 38835, 38836
	Cov. 45	38844	38842
	Cov. 50		38849
	Cov. 55		
	Cov. 60		

Table 5: List of SNV positions when the adaptive procedure of subsection Adaptive k-cMEC was activated for real Ashkenazim and simulated datasets of Chromosome 1. For each dataset, its row is identified by its average coverage (Avg. Cov.). The positions in column '4 to 7' are those for which the number of errors to correct was increased from 4 to 7, and similarly for the column '5 to 8'.