DCHap: A divide-and-conquer haplotype phasing algorithm for third-generation sequences

Yanbo Li and Yu Lin

Abstract—The development of DNA sequencing technologies makes it possible to obtain reads originated from both copies of a chromosome (two parental chromosomes, or haplotypes) of a single individual. Reconstruction of both haplotypes (i.e. haplotype phasing) plays a crucial role in genetic analysis and provides relationship information between genetic variation and disease susceptibility. With the emerging third-generation sequencing technologies, most existing approaches for haplotype phasing suffer from performance issues to handle long and error-prone reads. We develop a divide-and-conquer algorithm, DCHap, to phase haplotypes using third-generation reads. We benchmark DCHap against three state-of-the-art phasing tools on both PacBio SMRT data and ONT Nanopore data. The experimental results show that DCHap generates more accurate or comparable results (measured by the switch errors) while being scalable for higher coverage and longer reads. DCHap is a fast and accurate algorithm for haplotype phasing using third-generation sequencing data. As the third-generation sequencing platforms continue improving on their throughput and read lengths, accurate and scalable tools like DCHap are important to improve haplotype phasing from the advances of sequencing technologies. The source code is freely available at https://github.com/yanboANU/Haplotype-phasing.

Index Terms—haplotype phasing,third-generation sequencing, divide-and-conquer.

1 Introduction

The human genome is diploid; each chromosome has two copies: the maternal copy and the paternal copy. These two copies are highly similar and also exhibit different sites called variations. Single nucleotide polymorphisms (SNPs) are the most common variations between these two copies. The sequence of SNPs on a chromosome is usually called a haplotype. Haplotype information is important for genomics research, e.g., genetic variation is associated with the susceptibility to a wide variety of common disease [1]; the relationship of haplotypes between different generations is helpful to understand human history and evolution [2], [3].

The problem of haplotype phasing is to obtain SNP sequence on each chromosome from sequencing reads and a reference genome [4]. The rapid development of sequencing technologies provides opportunities to reconstruct haplotypes from sequencing reads. Since 2005, the next-generation sequencing (NGS) technologies have offered high-throughput, cost-effective sequencing platforms and thus revolutionized the genomics studies. While NGS sequencing data is being collected in enormous amounts, it still suffers from short lengths in resolving haplotypes: (1) short reads are prone to misalignment due to repeats in the genome, resulting in ambiguities for SNP calling, (2) reads are too short to span over multiple SNPs, resulting in discontinuity of reconstructed haplotypes. More recently, the third-generation sequencing (TGS) technologies, such as PacBio SMRT and ONT Nanopore, allow direct sequencing of single DNA molecules and generating long reads which are orders of magnitude longer than NGS reads (e.g., Illumina reads). Long TGS reads increase alignment confidence and link distant SNPs, thus open new opportunities for more complete and accurate haplotype phasing [5], [6], [7].

The problem of haplotype phasing is trivial when reads are error-free. However, this problem becomes NP-hard under most formulations for noisy reads [8], [9]. Therefore, various heuristic algorithms have been proposed to attack this problem. For example, FastHare sorts input reads by their starting positions, iteratively recruits one read at a time and greedily reconstructs two haplotypes [10]. Although FastHare runs linearly with respect to the problem input, it becomes inaccurate when dealing with inaccurate reads [10]. ReFHap builds a graph of reads, assigns a weight to a pair of reads based on their common SNPs and finally builds haplotypes consistent with the max-cut in the graph [11]. ReFHap scales quadratically with the number of reads, and thus becomes impractical to handle datasets with high read coverage. Several different dynamic programming algorithms (including ProbHap [12], WhatsHap [13] and others [14], [15]) have also been proposed to assemble the haplotypes with efficient enumeration strategies. Moreover, HapCUT and HapCUT2 are graph-based algorithms which compute max-cuts in read-haplotype graphs and thus find the corresponding haplotypes [5], [16]. These dynamic programming or graph-based approaches largely improve the accuracy of haplotype phasing, but still suffer from high time complexity with respect to either the maximum coverage [12], [13], [15] or the maximum number of SNPs in a single read [5], [14], [16].

In this paper we propose a divide-and-conquer algorithm, DCHap, which derives robust and accurate phas-

• *Yanbo Li and Yu Lin are with the Research School of Computer Science, Australian National University, 2601, Canberra, ACT, Australia. E-mail: {yanbo.li,yu.lin}@anu.edu.au*

Manuscript received XXX; revised XXX.

ing results for third-generation sequencing data. Our algorithm first identifies reliable regions and resolves the corresponding short haplotype segments, then uses reads to bridge these reliable segments to derive long haplotypes. Experimental results show that our DCHap approach is scalable to process high coverage error-prone long reads and achieves comparable or better performance for both simulated and real TGS data (including PacBio SMRT data and ONT Nanopore data.)

2 METHODS

2.1 Problem Formulation

The problem addressed in this paper is to reconstruct two haplotypes (H_1 and H_2) of an individual through sequencing reads. The common strategy is to align these reads into a reference genome, remove non-SNPs columns and encode heterozygous SNPs to build an input alignment matrix M. Each row of the m-by-n matrix M represents a read and each column represents a heterozygous SNP, where m is the number of reads and n is the number of SNPs. Each element $M[i,j]$ in M is in the alphabet $\{0,1,-\}$, where $M[i,j] = 0/1/-$ means the read R_i contains the major/minor/ambiguous allele at the j-th SNP position. Therefore, the problem of haplotype phasing is to reconstruct two haplotypes from the matrix M. As all columns correspond to heterozygous SNPs, two reconstructed haplotypes are both binary sequences and are complement to each other.

If reads are error-free, it is trivial to perfectly bi-partition rows of the matrix M into two groups such that every column in each group does not contain conflicting alleles (i.e., 0 vs 1) and the consensus sequences for two groups are naturally two haplotypes. For example, if we only consider four reads over four heterozygous SNPs, R_1('111$-$'), R_2('0001'), R_3('$-$001') and R_4('$-$110'), the perfect bi-partition is ($\{R_1$, $R_4\}$ and $\{R_2$, $R_3\}$) while two haplotypes $H_1 = 1110$ and $H_2 = 0001$, respectively. However, this problem becomes challenging for noisy NGS or TGS reads and various optimization measures (such as Minimum Fragment Removal, Minimum SNP Removal, Longest Haplotype Reconstruction [4] and Minimum Error Correction (MEC) [17]) have been proposed to handle erroneous reads. However, this problem is NP-hard under the above formulations for erroneous reads [8], [9]. In this paper, we focus on the most widely-used measure, Minimum Error Correction (MEC), *i.e.*, haplotype phasing is to find two haplotypes H_1 and H_2 with the minimum number of errors to be corrected in order to assign each read to either H_1 or H_2 perfectly. In other words, MEC represents the minimum number of elements in M that need to be flipped from 0 to 1 or vice versa to derive the perfect bi-partition and the pair of haplotypes H_1 and H_2. For example, if we consider the following four reads over four heterozygous SNPs, R_1('1100'), R_2('0101'), R_3('0001') and R_4('1110'), the MEC is 2 as we need to flip the third position in R_1 and the second position in R_2 to make a perfect bi-partition ($\{R_1$, $R_4\}$ and $\{R_2$, $R_3\}$) with two haplotypes $H_1 = 1110$ and $H_2 = 0001$.

2.2 Observation and Motivation

Under the Minimum Error Correction (MEC) measure, there are two naive and exponential-time algorithms to reconstruct a pair of optimal haplotypes. Note the any pair of haplotypes are two bit-complement binary strings and thus only one optimal haplotype needs to be reconstructed.

The first naive approach is to enumerate all possible binary haplotypes (2^n possible haplotypes where n is the number of SNPs) and to assign reads accordingly. The optimal haplotype will be the one with the minimum MEC. The second naive approach is to enumerate all possible bi-partitions (2^m possible partitions, m is the number of reads) for reads. Once the reads bi-partitions are fixed, the haplotype and its MEC can be inferred by consensus voting at each SNP position accordingly. The optimal haplotype with the minimum MEC can be derived after enumerating all reads bi-partitions.

Previous algorithms for haplotype phasing improve the enumeration behind these two naive algorithms. For example, instead of enumerating all possible full-length haplotypes or all bi-partitions of reads, dynamic programming approaches either enumerate length-restricted binary strings started at each column, where the enumeration length is the maximum number of SNPs in a read [14], or enumerate coverage-restricted reads bi-partitions at each column, where only reads covering the current column are considered to be partitioned [12], [13], [15].

As the global or local enumeration dominates the running time for existing algorithms, we would like to reduce the number of enumerations to improve the speed. In fact, in many genomic regions, the optimal haplotypes (or the bi-partitions of reads) are non-ambiguous and thus enumeration may not be needed. For example in Fig. 1 (a) columns 6-9, there are four reads containing non-ambiguous SNPs for these columns: R_5('1111'), R_6('1111'), R_7('0000') and R_8('0000'). As '1111' and '0000' are bit-complement, the optimal haplotype segments in this region are probably '1111' and '0000' while $\{R_5, R_6\}$ and $\{R_7, R_8\}$ are most likely the corresponding bi-partition. If we can identify such regions (similar to the sketches in [6]) in advance, it is possible to speedup the enumeration process by fixing the haplotype segments and the bi-partitions of reads in such regions. Therefore, we design a divide-and-conquer algorithm for our DCHap algorithm in the following three steps: (1) identify reliable regions; (2) bridge reliable regions; (3) post-processing. Refer to Fig. 1 for the pipeline of DCHap. We will explain each step of DCHap in the following sections.

2.3 Identify Reliable Regions

The basic idea of identifying reliable regions is to find consecutive SNPs regions resulting in corresponding haplotype segments and bi-partition of the covering reads. A fixed-length sliding window is used to scan the input alignment matrix from left to right. For each window, if a read contains only non-ambiguous alleles (i.e., only '0' and '1', without '$-$') in a window, the binary segment of the read in this window is called a *candidate window segment*. A window is called *reliable* if two most frequent *candidate window segments* are bit-complement. Two most frequent *candidate window segments* of a reliable window are called

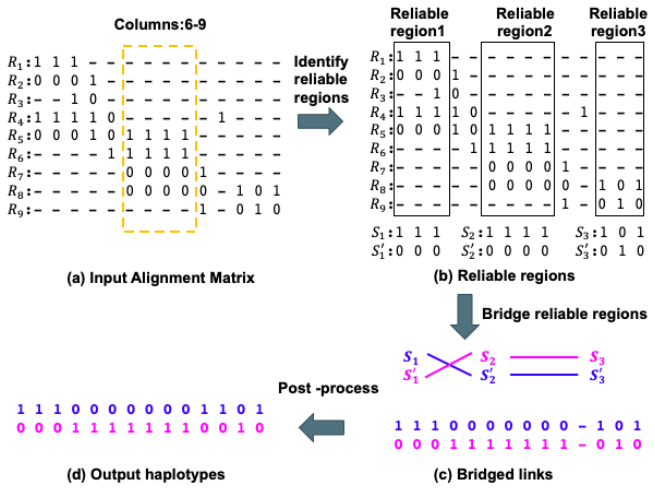

Fig. 1. The pipeline of DCHap. (a) Take the alignment matrix as the input. (b) Identify reliable regions using length-3 sliding windows. (c) Bridge neighboring reliable regions by finding links with the minimum MEC. (d) Post-process by using all reads to polish phased haplotypes.

window segments. For example in Fig. 1 (a), the first three columns form a length-3 *reliable* window because two most frequent candidate window segments ('000' and '111') are bit-complement [1]. Therefore, '000' and '111' become the window segments for this reliable window. Contiguous reliable windows may be merged into *reliable regions* if the corresponding window segments are consistent with respect to their overlaps. For example, the Reliable Region 2 in Fig. 1 (b) (columns 6-9) is derived by merging two length-3 reliable windows covering columns 6-8 and columns 7-9, respectively.

However, in certain genomic regions, especially with a low coverage, reads may be all sampled from just one haplotype. In this case, only one dominant candidate window segment is observed for each window, resulting in no reliable window in a long genomic region. To solve this issue, we apply an additional rule to find more reliable windows. This additional rule identifies more reliable windows where the most frequent candidate window segment appears at least twice and accounts for at least two thirds of all candidate window segments. It is worth noting that DCHap applies this additional rule only when no reliable window is found in a long genomic region (i.e., a region covering at least 10 SNPs in our setting).

It is worth mentioning that most genomic regions are reliable even for error-prone TGS reads. In two real datasets, 46x PacBio SMRT dataset [18] and 37x ONT Nanopore dataset [19] from same individual NA12878, reliable regions cover more than 90% of the whole genome and the phasing accuracy of reliable regions is above 99.9% (when the window size $= 3$). Moreover, gaps between two neighboring reliable regions are also very short (in average less than one[2], see Fig. 5 for the length distribution of two neighboring reliable regions for these two real datasets). Now we move on to show how DCHap bridges the gaps between reliable regions.

1. Note that '−−1' and '−−−' are ignored in counting candidate window segments as they contain ambiguous alleles.

2. Note that the minimum gap length is 0 when two neighboring reliable regions are next to each other (e.g., there is a gap of length 0 between two reliable regions covering positions 1-3 and positions 4-6, respectively).

2.4 Bridge Reliable Regions

For reliable region i, there are a pair of bit-complement segments, S_i and S_i'. For two neighboring reliable regions, i and $i+1$, DCHap needs to connect S_i to either S_{i+1} or S_{i+1}' and phase the gap alleles between reliable regions i and $(i+1)$ if it is possible.

For example, in Fig. 2 (top right), DCHap enumerates 32 possible links (5 binary bits) to bridge two reliable regions, including two possible connections between two reliable segments (1 binary bit) and four possible gap alleles (4 binary bits).

DCHap computes a MEC for each enumerated link with respect to the reads spanning over the gap alleles and record the link(s) with the minimum MEC. Three cases are summarized in Fig. 2 (bottom) to interpret such link(s) to bridge two reliable regions. Case (i) shows an ideal situation in which there is only one link with the minimum MEC and thus two reliable regions will be bridged. Case (ii) shows more than one links achieve the minimum MEC and these links only differ at local sites (*site-ambiguous*), thus DCHap will bridge two reliable regions and just set ambitious symbols at conflicting sites. For example, Fig. 2 (bottom middle) shows that two links, $S_1 0000 S_2'$ and $S_1 0010 S_2'$, only differ at the third gap site and thus $S_1 00{-}0 S_2'$ will be the output. Case (iii) shows more than one links achieve the minimum MEC and these links differ not only at gap sites but also the connection between segments (*link-ambiguous*), thus DCHap will not bridge these two reliable regions to avoid possible switch errors. For example, Fig. 2 (bottom right) shows that two links, $S_1 0000 S_2$ and $S_1 0011 S_2'$, result in different connection between segments (S_1 to S_2 *v.s.* S_1 to S_2') and thus $S_1 00$ and $11 S_2'$ will be the output.

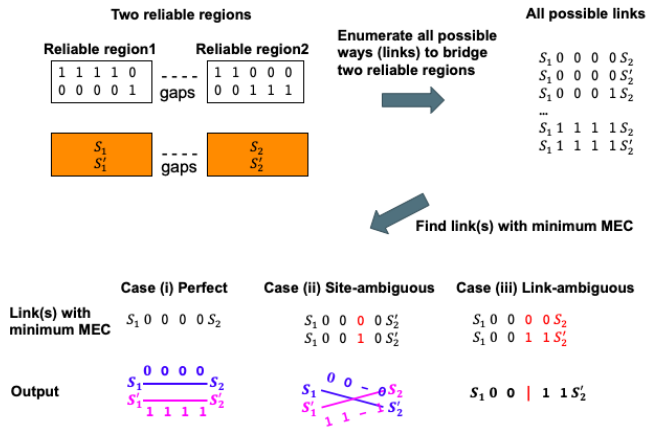

Fig. 2. The process of bridging two neighboring reliable regions in DCHap. Enumerate all possible ways (links) to bridge two reliable regions, find link(s) with the minimum MEC and output the phased haplotype segments.

Now we show how DCHap detects site-ambiguous and link-ambiguous cases. Assume that $\mathcal{L} = \{L_1, L_2, \ldots, L_m\}$ is the set of m links with the same minimum MEC. Without loss of generality, DCHap also binary encodes the choice of

the segment in a link and now each link L_i is represented by a binary string. Two positions j_1 and j_2 are *compatible* with respect to $\mathcal{L}$ if only one binary string (e.g., 01) or a pair of bit-complement binary strings (e.g., 01 and 10) are observed at positions j_1 and j_2 in $\mathcal{L}$. For example, in Case (ii) of Fig. 2, the first two gap positions are compatible (because only 00 is observed) while the last two gap positions are not compatible (because both 00 and 10 are observed). If a position is not compatible with any other positions (e.g., the third gap position in Case (ii) of Fig. 2), this position is *site-ambiguous* with respect to $\mathcal{L}$ and will be set as an ambiguous allele in the output. Positions that are compatible with each other form a *compatible group*. For example, the first three positions and the last three positions of Case (iii) in Fig. 2 form two compatible groups, respectively. Note that each compatible group with at least two positions admits a unique consensus segment. A *link-ambiguous* case appears between two compatible groups and will break the link between two reliable regions (e.g., a break between the first three positions and the last three positions of Case (iii) in Fig. 2).

As shown above, the problem of bridging reliable regions is to identify all compatible groups from the link(s) with the minimum MEC. DCHap checks pairwise compatibility to find all compatible groups and thus reconstruct consensus haplotypes when multiple links have the same minimum MEC.

While the example in the Fig. 2 covers the case to bridge two neighboring reliable regions, in fact, DCHap may bridge multiple reliable regions at a time. Enumerating all possible links among reliable regions can be done efficiently because reliable regions typically cover more than 90% of SNP sites (e.g., in both PacBio and ONT Nanpore real data of NA12878) and gaps between neighboring reliable regions are shown to be very short in most cases (refer to Fig. 5). In our experiments, we restrict the enumeration up to 16 binary bits and up to 32 SNP sites (including the SNPs in the reliable regions). As of long gaps, DCHap will enumerate bi-partitions of reads if the number of possible bi-partitions of reads is smaller than the number of all possible binary links.

2.5 Post-Processing

In DCHap, reliable regions are reconstructed only by reads covering consecutive non-ambiguous SNPs and reliable regions are bridged only by reads spanning over the gaps. To make use of all the reads and further polish the resulting haplotypes (similar to ReFHap [11] and ProbHap [12]), DCHap applies a post-processing step as follows:

$$C_{j,0} = \{i|(R_i \in A \bigcap R_{ij} = 0) \bigcup (R_i \in B \bigcap R_{ij} = 1)\}$$

$$C_{j,1} = \{i|(R_i \in A \bigcap R_{ij} = 1) \bigcup (R_i \in B \bigcap R_{ij} = 0)\}$$

$$h_j^A = \{\, 0\, if |C_{j,0}| > |C_{j,1}| 1 if |C_{j,0}| < |C_{j,1}| - otherwise;$$

R_i is a read, R_{ij} is the value of R_i at column j. A and B are the bi-partition of reads by assigning reads to current haplotype segments prior to post-processing. h_j^A is the haplotype at position j for the reads partition A. This post-processing strategy performs well in low-coverage data set, especially for phasing haplotype with extra-long and error-prone ONT Nanopore reads [19].

3 Results

3.1 Benchmarking Summary

Software tools: We benchmarked DCHap with three state-of-the-art phasing tools: HapCUT2 [5], FastHare [10], and WhatsHap [7], [13]. HapCUT2 and WhatsHap were downloaded from the author's website; FastHare we tested was implemented by [11]. ReFHap [11] and ProbHap [12] were excluded in the benchmarking because they were shown to suffer from time/memory issues when comparing to HapCUT2 [5].

Datasets: We used three different types of long sequencing reads: simulated reads (using PBSIM [20] to access the robustness against different error rates, read lengths and coverages), real PacBio SMRT data [18] and real ONT Nanopore data [19]. Two real datasets of PacBio SMRT and ONT Nanopore are from the same individual NA12878 [18], [19], for which we directly downloaded the alignment files and combined them with a set of heterozygous SNPs to build the input alignment matrix. Following the same procedure in HapCUT2 [5], the set of heterozygous SNPs positions for NA12878 is available from 1000 Genomes Project and the trio-phased haplotypes for NA12878 from 1000 Genomes Project [21] is set as the ground-truth.

Evaluation metrics: We used four metrics to evaluate the performance. The *switch error* and *mismatch error* are common measures for evaluating the phasing accuracy [5], [12]. A switch error indicates a phase discordant between two adjacent phased segments (of length 2 or more) when comparing with the ground-truth (e.g., there is a switch error between third and fourth position in MMMFF, where M and F represent the alleles on H_1 and H_2 respectively). A mismatch error indicates a phase discordant with respect to a single position (e.g, there is a mismatch error at third position in MMFMM). Note that a switch error is more serious than a mismatch error, because a switch error affects the global haplotype structures while a mismatch error only introduces certain local noise.

The AN50 evaluates the contiguity of assembled haplotype: it represents a value (a haplotype segment length in base pair) such that half of all phased SNPs are in those segments whose lengths longer than this value [22]. The phasing rate measures the completeness, which is defined as the number of SNPs in phased segments (of length two or more), divided by the total number of SNPs [12].

3.2 Simulated Data

We use PBSIM [20] to simulate PacBio SMRT reads from phased haplotypes for Chromosome 15 of NA12878 from 1000 Genomes Project [21]. A BAM file is derived by aligning these simulated reads to the Human reference genome hg19 using BWA-MEM [23]. In the preprocessing, WhatsHap [7], [13] uses SAMtools [24] to index the above BAM file while other software tools use ExtractHAIRS [5] to derive the input alignment matrix from the above BAM file.

We perform extensive experiments to assess the performance dependence on the coverage, read length and read

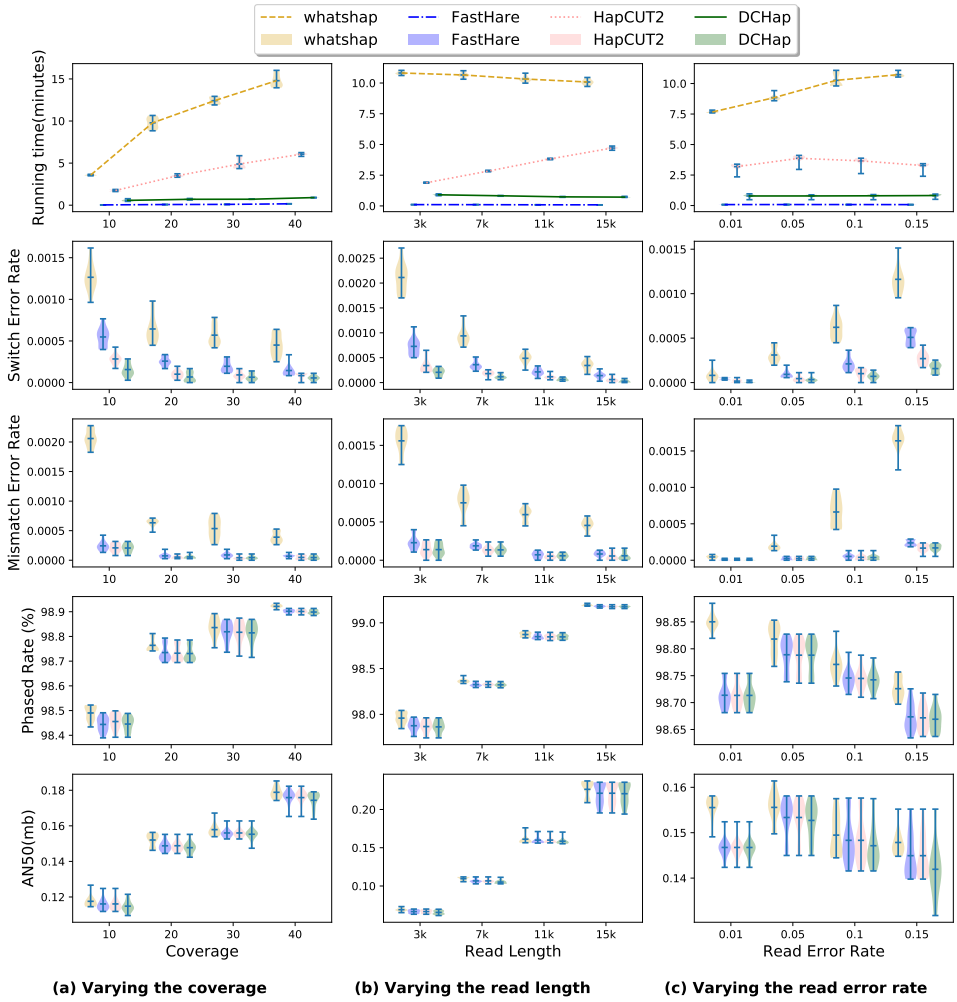

Fig. 3. The phasing benchmark on simulated reads using PBSIM [20] with different coverage depths, read lengths and read error rates. (a) the coverage varying from 10x to 40x when the average read error rate is 0.1 and the average read length is 10k. (b) the average read length varying from 3k to 15k when the read average error rate is 0.1 and the coverage is 20x. (c) the average read error rate varying from 0.01 to 0.15 when the coverage is 20x and the average read length is 10k.

error rate. For each parameter setting, 10 replicates are produced and tested to derive the average result. Among many parameter values tested, we show the following representative settings:

- Varying the coverage: the coverage varying from 10x to 40x when the average read error rate is 0.1 and the average read length is 10k. Refer to Fig. 3 (a).
- Varying the read length: the average read length varying from 3k to 15k when the error rate is 0.1 and the coverage is 20x. Refer to Fig. 3 (b).
- Varying the read error rate: the error rate varying from 0.01 to 0.15 when the coverage is 20x and the average read length is 10k. Refer to Fig. 3 (c).

Fig. 3 summarizes the phasing performance under the various parameter settings. The first row of Fig. 3 shows the running time[3]. FastHare and DCHap have a lower running time comparing to HapCUT2 and WhatsHap, and they also scale more efficiently with respect to the coverage, read length and error rate.

While FastHare runs fast because of its greedy strategy, DCHap benefits from the high percentage of reliable regions (above 90% for all cases) and the short average gap length (less than 2 for all cases) under various parameter combinations.

The second row of Fig. 3 shows that DCHap always achieves the lowest switch error rate, especially when the read error rate increases. The third row of Fig. 3 indicates that DCHap outperforms others with respect to the mismatch errors. The bottom two rows of Fig. 3 shows that WhatsHap achieves the highest phased rate and AN50 in all cases, however, it also comes with a higher switch error

3. We have excluded the prepossessing time, i.e., indexing using SAMtools [24] for WhatsHap [7], [13] and using ExtractHAIRS tool [5] for other software tools.

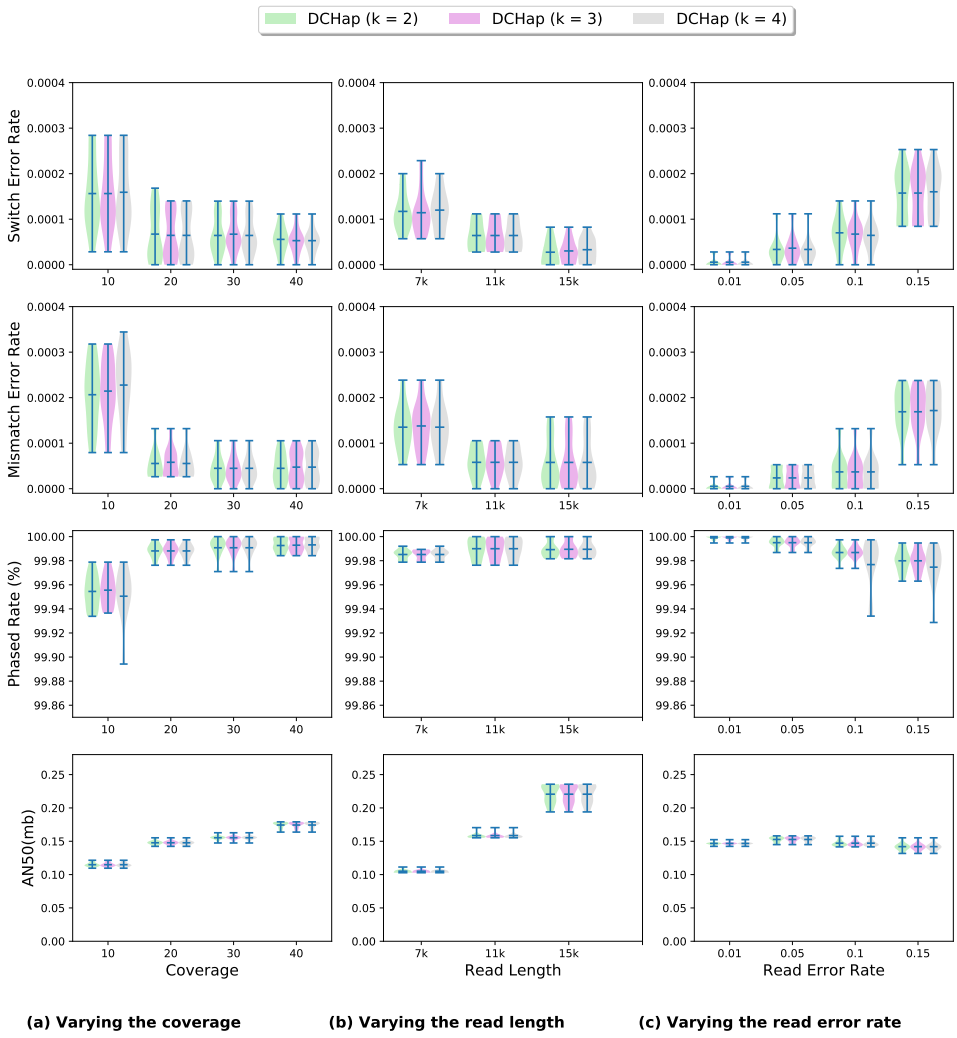

Fig. 4. Comparison on different lengths (k=2,3,4) of the sliding window in DCHap on simulated reads using PBSIM [20] with different coverage depths, read lengths and read error rates. (a) the coverage varying from 10x to 40x when the average read error rate is 0.1 and the average read length is 10k. (b) the average read length varying from 7k to 15k when the read average error rate is 0.1 and the coverage is 20x. (c) the average read error rate varying from 0.01 to 0.15 when the coverage is 20x and the average read length is 10k.

and mismatch error rates. DCHap has comparable phased rate and slightly shorter AN50 than HapCUT2 and FastHare because DCHap avoids connecting ambiguous links which may increase AN50 but also result in more switch errors.

In summary, although FastHare is fastest and WhatsHap phases the most number of SNPs in the above simulations, their phasing accuracy is still not satisfactory. DCHap and HapCUT2 achieve more robust and accurate results, but DCHap runs notably faster than HapCUT2 and the speedup increases when the coverage increases or the read length becomes longer in the third-generation sequencing data.

Choices of the sliding-window length in DCHap In the above experiments, DCHap sets the sliding-window length as 2. Now we vary the length of the sliding window from 2 to 4 in DCHap and summarize the results in the Figure 4. The phasing results are stable for DCHap for different window lengths. In general, the sliding window length is a trade-off between the running time and accuracy; a longer window length is preferred to handle more error-prone and longer reads while a shorter window length is more suitable to deal with less error-prone reads. By default, DCHap sets the sliding window length as 2 for PacBio datasets and 3 for Nanopore datasets, respectively.

3.3 Real PacBio SMRT Data

The PacBio SMRT reads from the individual NA12878 have a median depth of 46x and a median length of 2,650, and all the aligned reads to the human reference genome hg19 are downloaded from GIAB [18]. CrossMap [25] is then used to identify heterozygous SNPs positions from 1000 Genomes Project (based on hg18). The trio-phased haplotypes for NA12878 from 1000 Genomes Project [21] are used as the ground-truth.

Table 1 shows the performance of FastHare, HapCut2, WhatsHap and DCHap. FastHare is faster than other methods but also has the most switch errors and mismatch errors. While HapCut2 and WhatsHap largely improve on

TABLE 1
Comparison of phasing performance on the SMRT PacBio data (all chromosomes) [18]

Software	Switch Error Rate (#)	Mismatch Error Rate (#)	Phased Rate (%)	AN50 (k)	Time (minutes)
FastHare	0.00119 (1577)	0.00270 (3764)	99.050	209	13.02
HapCUT2	0.00096 (1278)	0.00265 (3701)	99.053	209	99.90
WhatsHap	0.00105 (1394)	0.00284 (3965)	99.136	214	211.47
DCHap	0.00084 (1121)	0.00264 (3680)	99.041	204	17.51

TABLE 2
Comparison of phasing performance on ONT Nanopore data (all chromosomes) [19]

Software	Switch Error Rate (#)	Mismatch Error Rate(#)	Phased Rate (%)	AN50 (k)	Time (minutes)
FastHare	0.00110 (1529)	0.00367 (5092)	98.778	3882	170.46
HapCUT2	0.00103 (1431)	0.00355 (4930)	98.767	3877	244.32
WhatsHap	0.00097 (1342)	0.00422 (5865)	98.961	3956	199.63
DCHap	0.00098 (1351)	0.00364 (5052)	98.776	3488	58.68

the phasing accuracy, their running time is significantly higher than FastHare. DCHap returns the minimum switch errors and mismatch errors, and its running time is still comparable to FastHare because the reliable regions cover more than 90% of the whole genome and the average gap lengths is less than one (refer to Fig. 5 (a) for the length distributions of gaps). DCHap's phased rate and AN50 are smaller than others because DCHap applies a conservative way to bridge reliable regions, and incorrect phasing links and positions may increase the phased rate and AN50.

3.4 Real ONT Nanopore Data

The ONT Nanopore reads from the individual NA12878 have a median depth of 37x and a median length of 5,950 [19]. Again, we use the trio-phased haplotypes for NA12878 from 1000 Genomes Project [21] as the ground-truth.

Table 2 summerizes the performance of FastHare, HapCUT2, WhatsHap and DCHap on this ONT Nanopore data. Interestingly, DCHap is faster than all other tools, including FastHare. As the running time of FastHare increases linearly with the average read length [5] and the extra long reads ($> 100k$) [19] in this dataset makes FastHare run slower than DCHap. DCHap achieves a 3-5X speedup on this dataset because the reliable regions cover 92.92% of the whole genome and the gaps between two neighboring reliable regions remain very short for error-prone nanopore reads (refer to Fig. 5 (b) for the distribution of gap lengths). DCHap are among the best with respect to both the switch errors and mismatch errors, while HapCUT2 has notably more switch errors and WhatsHap has significantly more mismatch errors.

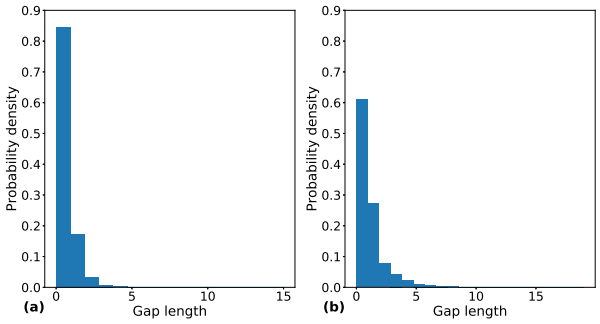

Fig. 5. Gap length distribution between neighboring reliable regions in DCHap. (a) is for 46x coverage SMRT PacBio data [18] and the (b) is for 37x coverage ONT Nanopore data [19].

4 Discussion and Conclusion

In this paper, we introduced DCHap, a fast and accurate algorithm for haplotype phasing using third-generation sequencing data. Extensive experiments on simulated and real data show that DCHap achieves a better or comparable phasing accuracy while offering scalability with respect to increasing read lengths and coverage. As the third-generation sequencing platforms continue improving on their throughput and read lengths, accurate and scalable tools like DCHap are important to improve haplotype phasing from the advances of sequencing technologies.

DCHap currently sets a fixed length of sliding windows to identify the reliable regions for the whole dataset. As we discuss in Results Section, different window lengths may be preferred for different read coverage and error rates. Therefore, choosing a dynamic window length may increase phasing performance as the read coverage varies along the chromosomes and the error rate changes with respect to different protocols in generating the TGS data.

Existing phasing tools (such as Probhap [12] and HapCUT2 [5]) use probabilistic models to maximize a likelihood function to phase haplotypes. Compared to the MEC model used in this paper, the flexibility in likelihood models allows to model different sources of errors (e.g., PacBio SMRT and ONT Nanopore) and integrate sequence data from diverse platforms [5]. How to extend DCHap to support a likelihood model is worth investigating.

5 Additional Files

The commands used in the benchmark are listed in the Additional file 1–Command Line.

References

[1] J. N. Hirschhorn, K. Lohmueller, E. Byrne, and K. Hirschhorn, "A comprehensive review of genetic association studies," *Genetics in medicine*, vol. 4, no. 2, p. 45, 2002.

[2] G. P. Consortium, "A map of human genome variation from population-scale sequencing," *Nature*, vol. 467, no. 7319, pp. 1061–1073, 2010.

[3] J. A. Hollenbach, G. Thomson, K. Cao, M. Fernandez-Vina, H. A. Erlich, T. L. Bugawan, C. Winkler, M. Winter, and W. Klitz, "Hla diversity, differentiation, and haplotype evolution in mesoamerican natives," *Human immunology*, vol. 62, no. 4, pp. 378–390, 2001.

[4] G. Lancia, V. Bafna, S. Istrail, R. Lippert, and R. Schwartz, "Snps problems, complexity, and algorithms," in *European symposium on algorithms*. Springer, 2001, pp. 182–193.

[5] P. Edge, V. Bafna, and V. Bansal, "Hapcut2: robust and accurate haplotype assembly for diverse sequencing technologies," *Genome research*, vol. 27, no. 5, pp. 801–812, 2017.

[6] F. Guo, D. Wang, and L. Wang, "Progressive approach for snp calling and haplotype assembly using single molecular sequencing data," *Bioinformatics*, vol. 34, no. 12, pp. 2012–2018, 2018.

[7] J. Ebler, M. Haukness, T. Pesout, T. Marschall, and B. Paten, "Haplotype-aware genotyping from noisy long reads," *BioRxiv*, p. 293944, 2018.

[8] R. Cilibrasi, L. Van Iersel, S. Kelk, and J. Tromp, "On the complexity of several haplotyping problems," in *International Workshop on Algorithms in Bioinformatics*. Springer, 2005, pp. 128–139.

[9] R. Schwartz *et al.*, "Theory and algorithms for the haplotype assembly problem," *Communications in Information & Systems*, vol. 10, no. 1, pp. 23–38, 2010.

[10] A. Panconesi and M. Sozio, "Fast hare: A fast heuristic for single individual snp haplotype reconstruction," in *International workshop on algorithms in bioinformatics*. Springer, 2004, pp. 266–277.

[11] J. Duitama, T. Huebsch, G. McEwen, E.-K. Suk, and M. R. Hoehe, "Refhap: a reliable and fast algorithm for single individual haplotyping," in *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*. ACM, 2010, pp. 160–169.

[12] V. Kuleshov, "Probabilistic single-individual haplotyping," *Bioinformatics*, vol. 30, no. 17, pp. i379–i385, 2014.

[13] M. Patterson, T. Marschall, N. Pisanti, L. Van Iersel, L. Stougie, G. W. Klau, and A. Schönhuth, "Whatshap: weighted haplotype assembly for future-generation sequencing reads," *Journal of Computational Biology*, vol. 22, no. 6, pp. 498–509, 2015.

[14] D. He, A. Choi, K. Pipatsrisawat, A. Darwiche, and E. Eskin, "Optimal algorithms for haplotype assembly from whole-genome sequence data," *Bioinformatics*, vol. 26, no. 12, pp. i183–i190, 2010.

[15] F. Deng, W. Cui, and L. Wang, "A highly accurate heuristic algorithm for the haplotype assembly problem," *BMC genomics*, vol. 14, no. 2, p. S2, 2013.

[16] V. Bansal and V. Bafna, "Hapcut: an efficient and accurate algorithm for the haplotype assembly problem," *Bioinformatics*, vol. 24, no. 16, pp. i153–i159, 2008.

[17] R. Lippert, R. Schwartz, G. Lancia, and S. Istrail, "Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem," *Briefings in bioinformatics*, vol. 3, no. 1, pp. 23–31, 2002.

[18] M. Pendleton, R. Sebra, A. W. C. Pang, A. Ummat, O. Franzen, T. Rausch, A. M. Stütz, W. Stedman, T. Anantharaman, A. Hastie *et al.*, "Assembly and diploid architecture of an individual human genome via single-molecule technologies," *Nature methods*, vol. 12, no. 8, p. 780, 2015.

[19] M. Jain, S. Koren, K. H. Miga, J. Quick, A. C. Rand, T. A. Sasani, J. R. Tyson, A. D. Beggs, A. T. Dilthey, I. T. Fiddes *et al.*, "Nanopore sequencing and assembly of a human genome with ultra-long reads," *Nature biotechnology*, vol. 36, no. 4, p. 338, 2018.

[20] Y. Ono, K. Asai, and M. Hamada, "Pbsim: Pacbio reads simulator—toward accurate genome assembly," *Bioinformatics*, vol. 29, no. 1, pp. 119–121, 2012.

[21] J. Duitama, G. K. McEwen, T. Huebsch, S. Palczewski, S. Schulz, K. Verstrepen, E.-K. Suk, and M. R. Hoehe, "Fosmid-based whole genome haplotyping of a hapmap trio child: evaluation of single individual haplotyping techniques," *Nucleic acids research*, vol. 40, no. 5, pp. 2041–2053, 2011.

[22] C. Lo, A. Bashir, V. Bansal, and V. Bafna, "Strobe sequence design for haplotype assembly," *BMC bioinformatics*, vol. 12, no. 1, p. S24, 2011.

[23] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with bwa-mem," *arXiv preprint arXiv:1303.3997*, 2013.

[24] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, and R. Durbin, "The sequence alignment/map format and samtools," *Bioinformatics*, vol. 25, no. 16, pp. 2078–2079, 2009.

[25] H. Zhao, Z. Sun, J. Wang, H. Huang, J.-P. Kocher, and L. Wang, "Crossmap: a versatile tool for coordinate conversion between genome assemblies," *Bioinformatics*, vol. 30, no. 7, pp. 1006–1007, 2013.

Yanbo Li received the MS degree on a topic related to genome assembly from Institute of Computing technology, Chinese Academy of Sciences. She is now working toward the Ph.D. degree in computer science at the Research School of Computer Science, Australian National University. She works on haplotype phasing and variants calling.

Yu Lin received the Ph.D. degree in computer science from EPFL, Lausanne, Switzerland. He was a postdoctoral scholar at the Department of Computer Science and Engineering, University of California, San Diego. Currently, he is a Lecturer at the Research School of Computer Science, Australian National University. His research interests include algorithm design and computational biology.