Majidian et al. BMC Bioinformatics (2020) 21:253
https://doi.org/10.1186/s12859-020-03584-5

BMC Bioinformatics

METHODOLOGY ARTICLE Open Access

Hap10: reconstructing accurate and long polyploid haplotypes using linked reads

Sina Majidian[1], Mohammad Hossein Kahaei[1*] and Dick de Ridder[2]

* Correspondence: kahaei@iust.ac.ir
[1]School of Electrical Engineering, Iran University of Science & Technology, Narmak, Tehran 16846-13114, Iran
Full list of author information is available at the end of the article

Abstract

Background: Haplotype information is essential for many genetic and genomic analyses, including genotype-phenotype associations in human, animals and plants. Haplotype assembly is a method for reconstructing haplotypes from DNA sequencing reads. By the advent of new sequencing technologies, new algorithms are needed to ensure long and accurate haplotypes. While a few linked-read haplotype assembly algorithms are available for diploid genomes, to the best of our knowledge, no algorithms have yet been proposed for polyploids specifically exploiting linked reads.

Results: The first haplotyping algorithm designed for linked reads generated from a polyploid genome is presented, built on a typical short-read haplotyping method, SDhaP. Using the input aligned reads and called variants, the haplotype-relevant information is extracted. Next, reads with the same barcodes are combined to produce molecule-specific fragments. Then, these fragments are clustered into strongly connected components which are then used as input of a haplotype assembly core in order to estimate accurate and long haplotypes.

Conclusions: Hap10 is a novel algorithm for haplotype assembly of polyploid genomes using linked reads. The performance of the algorithms is evaluated in a number of simulation scenarios and its applicability is demonstrated on a real dataset of sweet potato.

Keywords: DNA sequence analysis, Computational genetics, Haplotype, Synthetic long reads, Linked read, 10X genomics, Polyploid genomes, Clustering, Mathematical optimization

Background

Polyploids are organisms that possess three or more copies of each chromosome. There are numerous cases of polyploidy in the animal kingdom, including fish, amphibians and reptiles [1]. In plants, economically important crops such as potato, wheat, cotton and oat are polyploids [2]. For many genetic and genomic analyses, it is essential to know the sequence of alleles at variant sites corresponding to each homologous chromosome, i.e. the haplotypes. Haplotype information is needed to understand recombination patterns and uncover genotype-phenotype associations, with important applications in medicine [3] and plant breeding [2]. The development of

DNA sequencing technologies, specific protocols and computational tools make it possible to reconstruct the haplotypes of individuals to some extent. Nevertheless, obtaining haplotypes of polyploids remains a challenging computational problem [4].

Several algorithms for polyploid haplotyping have been developed in recent years for diploid and polyploid haplotyping [5, 6]. In the absence of DNA sequencing errors, the haplotyping problem reduces to a simple clustering if the provided reads are sufficiently long to cover neighbouring variants. If errors have to be taken into account, no polynomial-time solution is known. Therefore, different approximative and heuristic approaches have been used to estimate haplotypes. HapTree [7] is a greedy likelihood-based algorithm in which SNPs are added incrementally while keeping the tree of possible solutions to a manageable size. SDhaP [8] solves a correlation clustering problem using a gradient method to estimate the haplotypes. H-PoP [9], a heuristic algorithm, solves a combinatorial optimization problem called "polyploid balanced optimal partition". Another approach is to use the minimum fragment removal (MFR) model in which conflicting fragments (due to erroneous reads) are removed. Siragusa et al. devised a new algorithm based on the MFR model, which uses integer linear programming [10]. Polyphase, part of WhatsHap [11], is a method for polyploid haplotyping developed for short and long reads. Reads are clustered based on a position-based score, and haplotypes are threaded by dynamic programming. Poly-Harsh [12] is another method, minimizing the difference between the haplotypes and the input reads using a Gibbs sampling approach. The HapCompass algorithm [13] defines a SNP graph, removing a minimum number of weighted edges to obtain unique haplotypes. This is done by finding the spanning tree in such graph. RanBow [14], another program developed for short reads, first creates haplotype segments as the consensus sequences of fragments and then a graph in which haplotype segments and their overlaps are nodes resp. edges. The graph is used to merge the overlapping segments and calculate the haplotypeblocks [14]. For a recent review on different methods of polyploid haplotyping, see [6].

The above-mentioned algorithms are developed solely for short reads generated by Illumina DNA sequencing machines. These produce reads that have a low sequencing error rate (~ 0.1%) but do not provide long-range information, which is key in reconstruction of long haplotypes. Over the last years, a novel category of sequencing technology characterized by long-read sequencing was developed and commercialized by Pacific Biosciences and Oxford Nanopore [15]. However, successful application of long-read sequencing for haplotyping is hampered by the still high sequencing error rate and significant costs involved. Although a new technique has been recently been proposed to resolve the issue of high error rate [16].

Recently, 10X Genomics developed a linked-read sequencing library preparation strategy, commercialized through their Chromium platform, as a complementary technology to Illumina devices. This platform has the potential to provide long fragments at both low error rate and cost. In brief, the input genomic DNA, as little as 1 ng, is sheared into molecules of ~ 10–100 kbp. Subsequently, these molecules are isolated, partitioned into fragments, tagged with a unique 16 bp barcode, and amplified on beads in an emulsion. The resulting material is then sequenced by normal Illumina paired-end technology, which results in high-throughput reads that contain long-range genomic information through these barcodes [17]. The 10X technology described above is

one example of a general approach called synthetic long reads (SLRs), in which the low cost and high accuracy of short reads are combined with long range information provided by a barcoding scheme. Besides 10X Genomics, such technologies are commercialized by Illumina, Loop Genomics and Universal Sequencing Technology [18]. Such linked reads make it possible to assemble repetitive genomic regions as well as reconstruct long haplotype blocks. 10X Genomics delivers a likelihood-based algorithm in a software package called LongRanger to reconstruct haplotypes of diploid organisms such as humans [17, 19]. HapCUT2 [20] includes a program dedicated to linked-read haplotyping of diploids, which assembles the haplotypes to be maximally consistent with the read dataset by exploiting a likelihood-based model. Porubsky et al. proposed using a mixture of linked-read and strand-seq data to improve haplotype assembly [21].

However, no polyploid haplotyping algorithm is available at this moment, precluding the application of 10X-based haplotyping to a number of commercial crops and animals. Current polyploid haplotyping algorithms can be used on the obtained reads, ignoring the barcode information, but obviously the reconstructed haplotype blocks would be shorter than possible.

Exploiting the barcode information for haplotyping is possible by leveraging the so-called “fragment file” format. This format is used in preprocessing steps in several haplotyping algorithms [8, 20, 22]. The extractHAIRs (Extract HAplotype Informative Reads) program in the HapCUT2 package [20] can be used to produce a fragment file based on aligned reads and heterozygous SNPs. Such a file contains only the relevant information from reads: the coded alleles of each read at the SNP position and their quality (see Step 2 of "Hap++" Section and the illustrative example in Supplementary information: Figure S1). While extractHAIRs is dedicated to diploids and is used for haplotyping based on 10X linked reads, the same concept (with some modifications) may be applied to polyploids. Using the obtained fragment file as input of a haplotype assembly core, SDhaP [8], long haplotype blocks of a polyploid can be reconstructed. However, in our simulations for a small genome using the aforementioned approach we obtained poor results in terms of reconstruction rate and vector error rate. Moreover, SDhaP crashes for larger datasets. This indicates that this short-read haplotyping algorithm is currently unable to directly handle linked read data generated from a polyploid genome.

To tackle this computational problem, we designed Hap10 – a first haplotyping software package specifically tailored for 10X linked reads generated from a polyploid genome. We provide a general framework based on SDhaP that allows haplotyping at the chromosome scale. Furthermore, we propose a novel optimization method that generates more accurate haplotypes with almost the same block length.

Methods

We have developed the Hap10 package to reconstruct haplotypes from a polyploid genome using linked reads. Prior to haplotyping, several processing steps on sequencing reads are required. These include barcode handling, read alignment and variant calling, which are discussed in "Preparation procedure" Section. Thereafter, Hap++, a new pipeline for polyploid haplotyping of linked reads is explained in detail in "Hap++" Section. This pipeline uses SDhaP as the assembly core. Lastly, the Hap10 algorithm is presented in "Hap10: an improved assembly core" Section. This algorithm leverages the

Hap++ pipeline, supplemented with a novel optimization based on an augmented Langrangian formulation as the assembly core. "Experimental setup" Section concludes by discussing the data and performance measures used for validation of the method.

Preparation procedure

First, the 16 bp 10X barcode is removed from the beginning of each paired-end read generated by the Illumina device. The barcode is stored as a read tag for further use. The possibility of sequencing errors in the barcode calls for an error correction scheme based on the known set of barcodes. Next, the reads are aligned to the reference genome using the barcode information. The barcodes contain long range information that can help provide a better alignment, particularly in repetitive genomic regions. These steps are performed using the LongRanger package (version 2.2.2) [19] provided by 10X Genomics, which generates a binary sequence alignment (BAM) file in which the barcodes are stored in the BX tag of each read. Subsequently, single nucleotide polymorphism (SNP) sites and their genotypes are called using the FreeBayes package (version 1.3.1) [23] with "-p 3" and "-p 4" for triploids and tetraploids, respectively and stored as a variant call format (VCF) file. The pipeline is depicted in Fig. 1.

Hap++

Hap++ is a fast program to reconstruct haplotypes in polyploids by exploiting linked read information. It consists of three main steps:

1) extracting haplotype-relevant information from input BAM and VCF files;
2) extracting molecule-specific fragments;
3) extracting strongly connected components of fragments.

The output of the last step can then be used by SDhaP to assemble the haplotypes. The three steps are described below.

Step 1. Extracting haplotype information

We first extract data relevant for haplotyping from the BAM and VCF files. As only heterozygous SNPs are informative for haplotyping, we filter out the homozygous variants from the VCF file. Next, we remove reads that cover fewer than two SNPs, since these do not provide any information for haplotyping. Subsequently, we extract the alleles of SNP sites of each read stored in the BAM file. In order to exploit long-range information provided by the barcodes, we combine the obtained fragments originating from the same 10X bead, i.e. with the same barcode. This results in long barcode-specific fragments. If there are two mismatching alleles for a SNP site corresponding to a specific barcode, we choose the one with the higher base quality. The result is a compact fragment file, similar to the output file of extractHAIRS [20, 24].

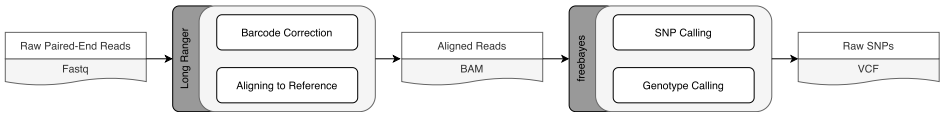

Fig. 1 Preparation procedure for haplotyping of linked read data: barcode correction, read alignment and SNP/genotype calling. The output consists of aligned reads (BAM file) and called variants (VCF file)

Step 2. Extracting molecule-specific fragments

The reads generated from the molecules in the 10X bead have identical barcodes. In an ideal case, the microfluidic device is expected to produce one molecule within each bead. In reality however there are, on average, 10 molecules per bead that originate randomly from one of the haploid chromosomes [17]. Therefore, the haplotypic origin of molecules with the same barcode is not identical, as discussed in [20]. As a result, parts of fragments in the fragment file are derived from different haplotypes, which misleads the haplotype assembly program. To tackle this issue, we propose a fragment processing scheme to extract molecule-specific fragments from each barcode-specific fragment. This is done by splitting barcode-specific fragments into several parts such that distant parts are retained as individual fragments. To this end, we use the mean-shift clustering algorithm [25] by means of its Python implementation from the Scikit-learn package [26]. We set the bandwidth of clustering to half of the expected 10X molecule length. This approach is based on the fact that molecule coverage is very low, and thus, molecules with the same barcode are generally distant from each other.

Step 3. Extracting strongly connected components of fragments

It is crucial to have a decent reference genome, because read alignment to the reference is upstream of haplotyping ("Preparation procedure" Section). However, in practice, reference genomes are incomplete and contain assembly gaps (usually represented by Ns). This affects haplotyping: if the reference contains a gap with length comparable with that of the 10X molecules, only few fragments connect the two sides of the gap and sequencing/mapping errors can have undue influence on the haplotyping process.

To prevent such problems, we first create a graph G in which fragments are considered as vertices $v \in G$. The weight w_{ij} of the edge $e_{ij} = (v_i, v_j)$ between two nodes is calculated as the number of shared SNPs between two corresponding fragments, inspired by SDhaP [8]. As a demonstration, we generated such a graph for a read dataset (depicted in Supplementary information: Figure S2), in which the length of the 10X DNA molecules is slightly higher than 50 kb, the length of the simulated gap. In this graph, one edge was found to connect two separate parts. This is based on a single molecule covering two distant SNPs in the vicinity of the gap. However, a single barcoded fragment is not enough for linking all haplotypes. Consequently, the accuracy of the whole haplotype block decreases.

As errors other than those due to gaps can lead to spurious edges in G, we provide a generic solution based on extracting strongly connected components of fragments. To this end, we exploit an iterative bipartitioning method based on the normalized cut (NC) [27]. We calculate the normalized Laplacian matrix (L_N) of G based on the corresponding weight matrix W:

$$L_N = D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}, \tag{1}$$

where D is the degree matrix of the graph, a diagonal matrix with $D_{ii} = \sum w_{ij}$. After calculating the eigenvalue decomposition of L_N, we use the eigenvector $(E_2)^j$ that corresponds to the second smallest eigenvalue in order to bipartition the graph. In [27], it is shown that minimization of NC value is equivalent to minimization of a Rayleigh quotient, $\frac{x^T L_N x}{x^T x}$. This can be used to show that the second eigenvalue presents the optimum

partition in terms of the NC value defined in (2) [27]. The sign of the elements of vector E_2 indicates the affiliation of fragments to either subgraph G_1 or G_2. Then, we calculate the NC value:

$$NC(G_1, G_2) = \frac{\sum_{i \in G_1, j \in G_2} w_{ij}}{\sum_{i \in G_1; \forall j} w_{ij}} + \frac{\sum_{i \in G_1, j \in G_2} w_{ij}}{\sum_{\forall i; j \in G_2} w_{ij}}. \quad (2)$$

If NC is greater than a pre-specified threshold t, we stop the bi-partitioning procedure; otherwise, we continue bi-partitioning for each remaining partition. We set t to 0.03 for all simulations throughout the paper. When this step is finished, we output all strongly connected components of fragments as individual fragment files for processing by the assembly core, SDhaP. The Hap++ pipeline is depicted in Fig. 2. Note that this pipeline can be parallelized; specifically, the assembly core can be run on each strongly connected fragment simultaneously.

Hap10: an improved assembly core

The Hap10 pipeline leverages the Hap++ pipeline and adds a novel optimization as the assembly core. The goal of a haplotype assembly algorithm is to reconstruct K haplotypes $H = \{h_1, ..., h_K\}$ from N aligned fragments $R = \{r_1, ..., r_N\}$ generated by DNA sequencing of a K-ploid organism. This definition is universal and applies to different sequencing data types. Each r_i is assumed to originate from a single haplotype, as is the case for Illumina reads. As we discussed earlier, in linked read technology, we use molecule-specific fragments as r_i in our pipeline.

As a basis for Hap10, we use the three-step approach introduced by SDhaP:

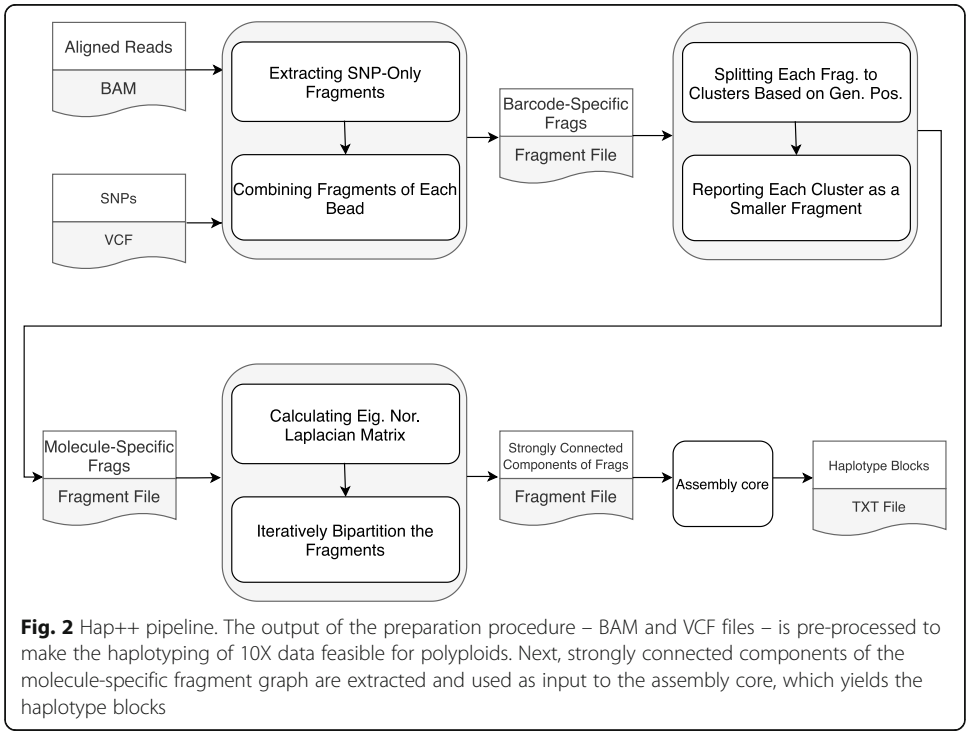

Fig. 2 Hap++ pipeline. The output of the preparation procedure – BAM and VCF files – is pre-processed to make the haplotyping of 10X data feasible for polyploids. Next, strongly connected components of the molecule-specific fragment graph are extracted and used as input to the assembly core, which yields the haplotype blocks

I. Construct a fragment graph (similar to that of "Hap++" Section, Step 2) with weights between fragments (vertices) i and j calculated as.

$$W_{ij} = \frac{\#\text{mismatched alleles} - \#\text{matched alleles}}{\#\text{shared SNPs}}. \tag{3}$$

II. Split the fragments into K clusters, exploiting the graph weights.
III. Combine fragments of each cluster into a single haplotype using majority voting.

The reconstructed haplotypes are reported in a text file in a format similar to HapCUT2's output [20] presented in Supplementary information: Table S1.

Here, we explore step II of the assembly core. We use max- K-cut modelling [28] for clustering the graph based on the edge weights W, which results in the following convex optimization problem over $X \in \mathbb{R}^{N\times N}$:

$$\min Tr(WX) \text{ s.t.} X_{ij} \geq -\frac{1}{k-1}, X \succcurlyeq 0, \tag{4}$$

in which $X \succcurlyeq 0$ indicates that X is a positive semi-definite matrix. Note that $\hat{X}_i$, the i-th column of the optimum $\hat{X}$, corresponds to the i-th fragment. The matrix $\hat{X}$ is used to estimate the cluster membership of each fragment using a randomized approach [28]. Each fragment is assigned to the k-th cluster when the corresponding column is the closest to the k-th random vector in terms of inner product [29]. To do so, firstly, K random vectors $\{v_1, ..., v_K\}$ are generated, each an $N \times 1$ vector with elements drawn from a standard normal distribution. Next, inner products between columns of $\hat{X}$ and these random vectors are calculated and the i-th fragment is assigned to the k-th cluster, corresponding to the k-th haplotype, if

$$k = \text{argmax} \left\{ \left\langle \hat{X}_i, v_1 \right\rangle, ..., \left\langle \hat{X}_i, v_K \right\rangle \right\}, \tag{5}$$

in which $\langle ., . \rangle$ represents the inner product of two vectors.

We exploit dual theory in optimization to solve the semidefinite programming problem (3). Note that the identity matrix is a positive definite matrix, and all its elements are nonnegative. Thus, the identity matrix belongs to the interior of the optimization domain. Thus, the optimization is strictly feasible. Therefore, Slater's condition is satisfied for the optimization, which immediately results in strong duality (section 5.2.3 of [30]). To derive the dual optimization problem of (4), the Lagrangian function can be written as $L(X, \lambda, Z)=$

$$Tr(WX) + \sum_t \lambda_t \left(\frac{1}{K-1} - Tr\left(A_t X_{\{(i,j)|Ni+j=t\}}\right) \right) - Tr(ZX) \text{ s.t.} \lambda \geq 0, Z \succcurlyeq 0, \tag{6}$$

in which A_t is a matrix with the same dimensions as X of zeroes with a 1 in the (i, j)-th element. Then, (6) can be rearranged to

$$L(X,\lambda,Z)=\frac{1}{K-1}\sum_t \lambda_t + Tr\left(X\left(W-\sum \lambda_t A_t - Z\right)\right)\ \text{s.t.}\lambda\geq 0, Z\succcurlyeq 0. \tag{7}$$

Since the second term is affine in X, we should make it bounded. To this end, the weight of the affine function should be zero. Thus, the maximization (6) can be simplified to

$$\max \frac{1}{K-1}\sum_t \lambda_t\ \text{s.t.}\, W-\sum \lambda_t A_t - Z = 0, \lambda\geq 0, Z\succcurlyeq 0. \tag{8}$$

To achieve an unconstrained optimization, we define the augmented Lagrangian function of the optimization as [31]:

$$L_\mu(\lambda,Z,Y)=\frac{1}{K-1}\sum_t \lambda_t + Tr\left(Y\left(W-\sum \lambda_t A_t - Z\right)\right)+\frac{\mu}{2}\left\|W-\sum \lambda_t A_t - Z\right\|^2. \tag{9}$$

A novel iterative optimization scheme for solving the max- K-cut problem then becomes:

$$\begin{aligned}(\lambda^{i+1},Z^{i+1}) &= \text{argmax} L_\mu(\lambda,Z,Y^i)\ \text{s.t.}\lambda\geq 0, Z\succcurlyeq 0, Y^{i+1}\\ &= Y^i+\sigma_i\left(W-\sum \lambda_t^i A_t - Z^i\right)\end{aligned} \tag{10}$$

Then, the optimality condition of the first optimization results in a linear equation, which is solved by a Newton conjugate gradient approach (Section 10.2 of [32]). We stop the iteration when the relative duality gap (defined as $\frac{obj_p-obj_d}{1+obj_p+obj_d}$ in which obj_p and obj_d are the value of primal and dual objective functions, respectively [33]) falls below a certain convergence threshold, which we set to 0.01. Note that the smaller this threshold, the longer the runtime but the better the estimate (Supplementary information: Table S2). Then, the primal optimal point X is found using complementary slackness conditions (section 5.5.2 of [31]). To implement the mentioned algorithm, we use the SDPNAL+ package [33].

Experimental setup

Data

In order to evaluate the performance of the developed pipelines and algorithms, we consider numerous scenarios on both simulated and experimental data. First, we performed extensive simulation experiments using the reference genome of potato (*Solanum tuberosum)* as a basis. We first simulated data based on an arbitrarily selected region of one million base pairs (1 Mb) starting from position 5,032,020 on chromosome 1 and subsequently used the full chromosome 1 sequence (88.6 Mb). We introduce SNPs in the reference at a rate of one per 100 or 1000 (for the 1 MB region) and one per 100 for the full chromosome. We generate synthetic triploid and tetraploid genomes as FASTA files by combining $K = 3$ resp. $K = 4$ mutated copies of the reference sequence using the haplo-generator routine from the Haplosim package [4]. This package also produces K true haplotypes in a text file, including the genomic positions of SNPs and the corresponding alleles, which are used for evaluation (see "Performance assessment" Section).

Subsequently, we simulated several linked-read datasets following the 10X technical specifications, using the LRSIM package [34]. We set the number of molecules per bead (−m) as 10 and assigned the number of barcodes (−t) such that the molecule coverage is 0.2, as discussed in the 10X Genomics technical note (No. CG00044). The output of each LRSIM simulation consists of two FASTQ files, containing paired-end reads with length of 2×151 bp, in which the first 16 bases are the barcode sequence. The outer distance between the two reads in a pair is set to the default value, 350, with a standard deviation of 35. Then, as described in "Preparation procedure" Section, the LongRanger and FreeBayes packages are used for aligning reads and calling SNPs, respectively.

To the best of our knowledge, there is no publicly available, real dataset for a polyploid organism containing true haplotype sets, which makes it hard to determine accuracy. To obtain an impression of the distribution of haplotype block lengths and runtimes, we download 10X raw read data of hexaploid sweet potato (*Ipomoea batatas*) from the NCBI database (accession SRX4706082) [35].

Performance assessment

To evaluate the length of the reconstructed haplotypes, we calculate and report the mean value over all haplotype blocks. To assess the accuracy of each algorithm, we consider two criteria: **reconstruction rate**, a measure of local accuracy; and **vector error rate**, a more global measure. Given reconstructed haplotypes $\hat{H} = \{\hat{h}_1, \ldots, \hat{h}_K\}$ and ground truth haplotypes $H = \{h_1, \ldots, h_K\}$, the reconstruction rate is defined as:

$$RR = 1 - \frac{1}{kL} \min_p \sum_{k=1}^{K} D_H\left(\hat{h}_k, h_{p_k}\right), \tag{11}$$

in which L is the haplotype length and $D_H(.,.)$ is the Hamming distance function, which counts the number of mismatch elements between its arguments. Additionally, p is a permutation on the set $\{1, \ldots, K\}$, and p_k is the k-th element of p. We calculate this criterion for each haplotype block and report the average. The vector error rate is calculated by finding the minimum number of switches needed in haplotype segments in order to match $\hat{H}$ to H; this number is then divided by the haplotype length [9, 24].

Since for real data there is no ground truth for assessing the performance of the estimated haplotype, the mentioned metrics cannot be used. To handle this issue, another metric, the **Minimum Error Correction** (MEC) score, has been frequently used in the literature [7, 8]:

$$MEC\left(R, \hat{H}\right) = \sum_{i=1}^{N} \min_k D_{HE}\left(R_i, \hat{h}_k\right) \tag{12}$$

in which R_i is the i-th pre-processed read ("Preparation procedure" Section). For haplotypes with a length of l, the extended Hamming distance function is defined as $D_{HE}(R_i, \hat{h}_k) = \sum_{j=1}^{l} d(R_i(j), \hat{h}_k(j))$. The value $d(R_i(j), \hat{h}_k(j))$ will be one when read R_i covers the j-th position of haplotype $\hat{h}_k$ and both are of the same allele, and will be zero otherwise. To interpret this metric, we should note that MEC shows the extent of match between the reconstructed haplotypes and the read dataset.

Results

We have developed Hap10, a novel pipeline for haplotyping polyploids based on linked-read (SLR) data. The basis of Hap10 is a set of pre-processing steps called Hap++. After application of Hap++, SDhaP [8] can be used as an assembly core. We also propose an alternative core, based on the SDPNAL+ algorithm. The combination of Hap++ and the new assembly core is called Hap10.

To obtain an impression of the performance of SDhaP, Hap++/SDhaP and Hap10, we performed extensive simulations based on real-world data, the potato genome. This allows us to investigate accuracy ("Performance assessment" Section) and run time in different scenarios, varying sequence length, coverage, ploidy, heterozygosity etc. ("Simulated data" Section). We then apply the pipeline to real-world data to evaluate performance in terms of haplotype block length and run time ("Real data" Section).

Simulated data

We first applied the various algorithms on 10X data simulated based on a relatively short stretch of the potato genome, of 1 Mb ("Experimental setup" Section), to learn about the influence of various genome and sequencing characteristics.

Linked-read information yields longer haplotypes

As a first test, we applied SDhaP to the simulated read data with and without taking the barcode information into account. The program has no problem dealing with data for a region of this length. Without linked-read information, the reconstruction rate and the vector error rate are relatively good, but the reconstructed haplotype blocks are very short, 11.8 SNPs on average (Supplementary information: Table S3, first row) as is to be expected.

Taking the linked read information into account here improves average haplotype block length dramatically, to over 6000 SNPs (Supplementary information: Table S3, second row compared to the first row). At the same time, the reconstruction rate drops, and the vector error rate increases, indicating low quality haplotypes. This is due to the effect of mixed haplotypic origin of fragments, misleading the haplotype assembly program. It can be also considered the consequence of the poor connections between subgraphs, insufficient for haplotyping, as illustrated in Supplementary information: Figure S2. An approach in which haplotypes are calculated independently on three equally sized parts of the region of interest supports this: the average block length decreases, but both reconstruction rate and vector error rate improve (Supplementary information: Table S3, third row compared to the second row). This suggests that while SDhaP in principle works for haplotype assembly in polyploids, performance may be improved by pre-processing the data. From here on, all results reported for SDhaP are based on barcode information.

Preprocessing by hap++ yields shorter, more reliable haplotype blocks

To solve the problems encountered in "Simulated data" Section, we developed a novel preprocessing pipeline Hap++, to extract strongly connected components from the fragment graph. This reduces the potential for erroneous haplotype assembly, at the expense of a reduced haplotype block length. We apply Hap++ to triploid and tetraploid

data simulated on the 1 Mb region taken from the potato genome, at various levels of coverage (2, 5 and 10 per haploid) and different SNP rates (0.01 and 0.001). We repeated the simulations 5 times and report average haplotype block lengths, reconstruction rates and vector error rates in Fig. 3.

Hap++ indeed yields much shorter haplotype blocks (e.g. 339.9 versus 787.2 Kb for SDhaP for triploid, SNP rate 0.01, coverage 10), but drastically improves performance over SDhaP. The reconstruction rate increases, in particular for the triploid simulations, and the vector error rate drops to below 0.1 for almost all simulations where for SDhaP it can reach as high as 0.6. This indicates that the spurious connection problem

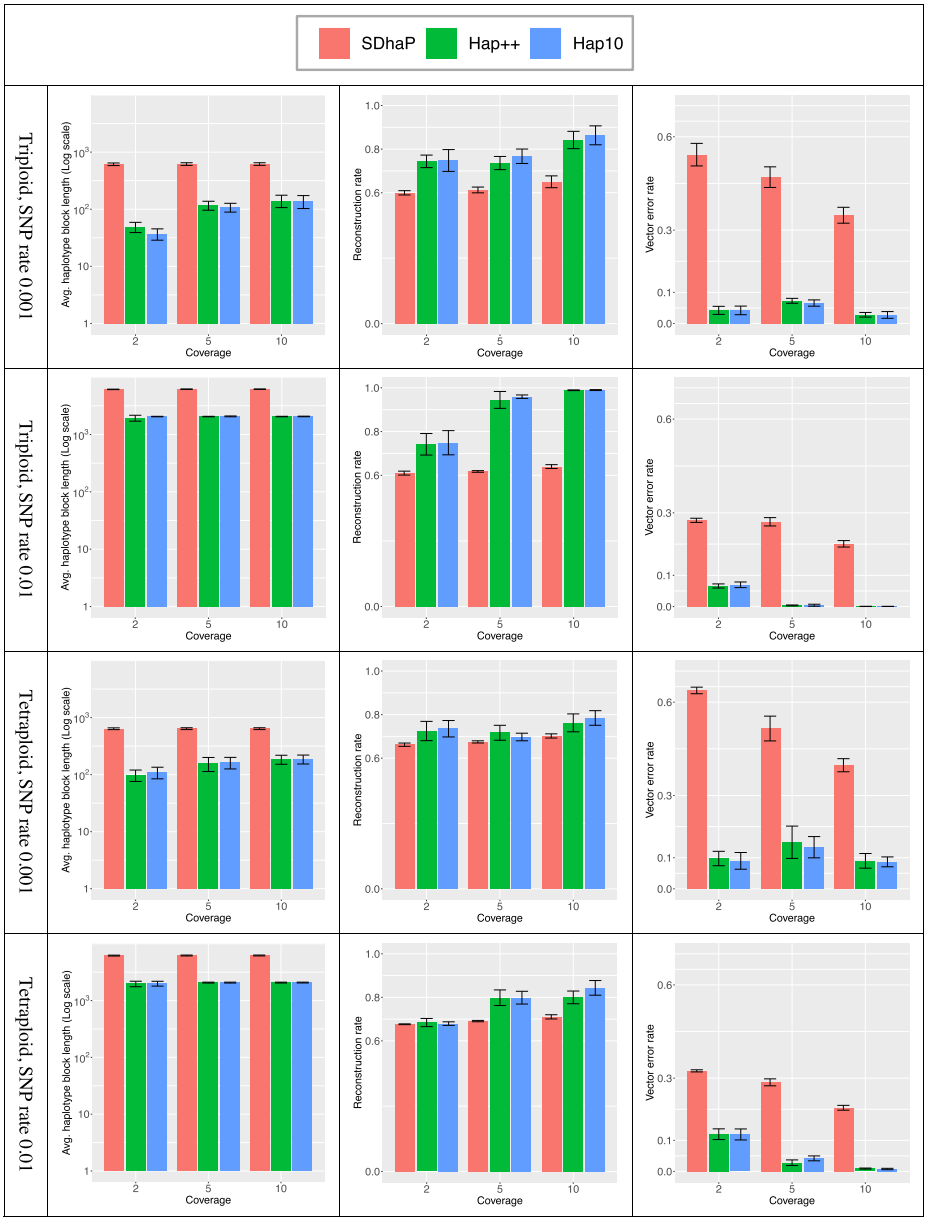

Fig. 3 Average haplotype block length (left column, note the logarithmic scale), reconstruction rate (middle) and vector error rate (right) for different coverage levels. Bars indicate averages, whiskers standard deviation of 5 repeated simulations

discussed before occurs in practice and seriously impacts results. It is clear that the SNP rate has a large influence on performance: at low SNP rates, average haplotype block lengths are shorter and accuracy is higher, again particularly for the triploid simulations.

Figure 3 also shows that performance improves with coverage (as expected), and that a coverage of 2 is so low that all methods make errors, due to the fact that SNPs often cannot even be detected. Hap++ benefits more quickly from increasing coverage than SDhaP. SDhaP performance improves up to a coverage of 10 per haploid and keeps improving, as spurious connections in the fragment graph will increasingly be supported by more connections and errors will be counteracted by solid data: in fact, SDhaP needs 5 times as much coverage to reach a similar vector error rate (Supplementary information: Table S4).

Figure 4 shows performance at different ploidy levels. While haplotype block length is invariant to the ploidy level, in most cases more trustworthy haplotypes are attained at higher ploidy levels. To understand this, note that the max-K-cut randomized approach (part of the assembly core) is theoretically guaranteed to converge to near the optimal value (by a factor of $(1-\frac{1}{K}+\frac{2\ln K}{K^2})$, a function increasing in K, as presented in Theorem 1 of [28]). However, limited precision in the SDP solver means this solution is not always found in practice.

Hap++ deals better with imperfect 10X data

Ideally, the 10X technology ensures each unique barcode is assigned to fragments that originate from a single, long DNA molecule. In practice however, fragmentation is imperfect, leading to shorter molecules, and more than one molecule may receive the same barcode (see "Hap++" Section). Hap++ contains a pre-processing step to cluster reads based on the expected molecule size, to avoid the concatenation of different molecules in a single line of the fragment file as much as possible.

Figure 5 (top) shows performance as a function of both the number of molecules that on average receives the same barcode (in simulated data). The difference between SDhaP and Hap++ is striking, in that vector error rate increases drastically with the number of molecules per barcode for SDhaP but remains negligible for Hap++. The

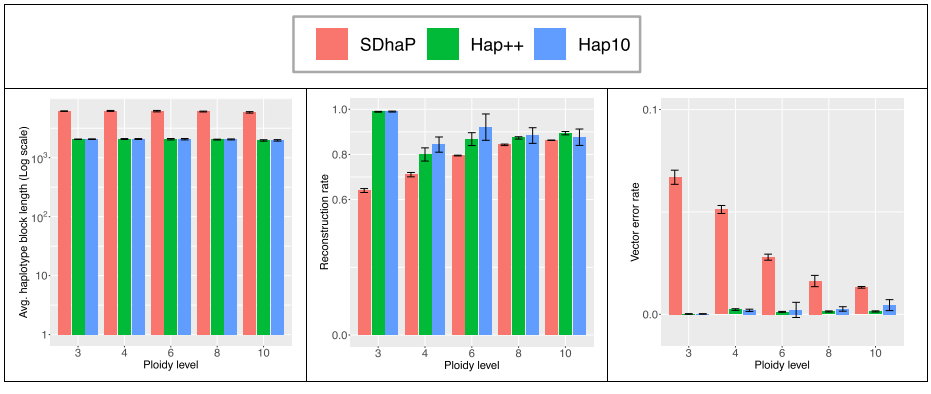

Fig. 4 Average haplotype block length (left), reconstruction rate (middle) and vector error rate (right) for different ploidy levels (SNP rate 0.01). Bars indicate averages, whiskers standard deviation of 5 repeated simulations

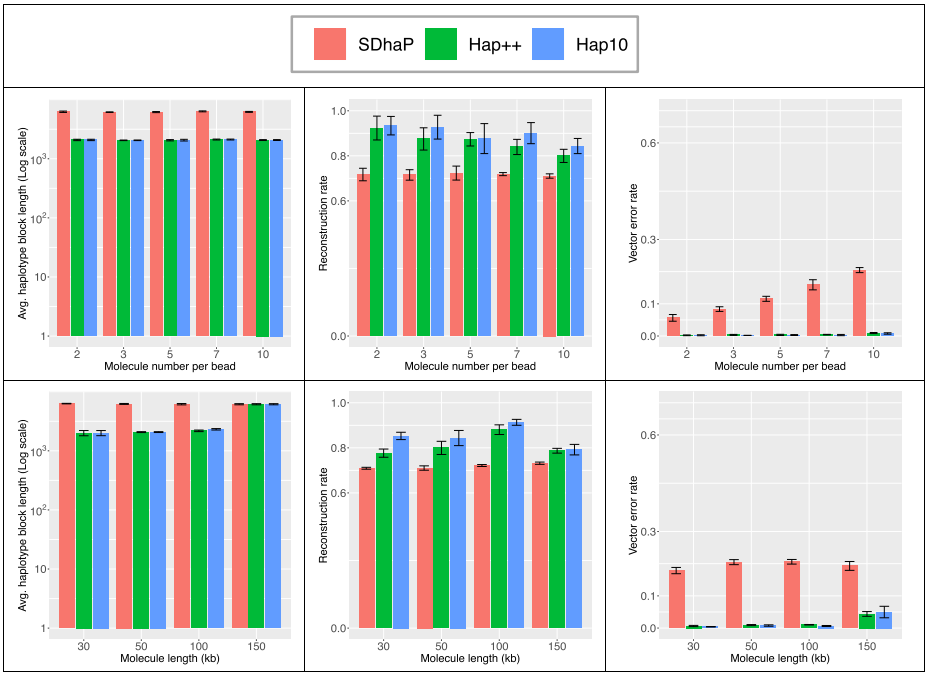

Fig. 5 Average haplotype block length (left), reconstruction rate (middle) and vector error rate (right) for different settings of the 10X linked-read simulation (SNP rate 0.01, tetraploid), varying the number of molecules per bead (top) and the molecule length (bottom). Bars indicate averages, whiskers standard deviation of 5 repeated simulations

Hap++ reconstruction rate decreases somewhat, but remains higher than that of SDhaP up to at least 10 molecules per barcode – which, given the sequence length of 1 Mb and the molecule length of 100 kb entails a significant probability of overlap between molecules with the same barcode.

We also varied the length of the 10X molecules in the simulations, from 30, 50 and 100 to 150 kb. Figure 5 (bottom) shows that longer molecules yield better haplotypes in terms of reconstruction rate due to the improved long-range information, but eventually increases the vector error rate, likely due to the increased probability of overlap of such long molecules (150 kb in a 1 Mb region).

Hap10 improves performance, at considerable computational cost

Figure 3 also includes performance of Hap10, a combination of the Hap++ pre-processing stage with a new assembly core based on the SDPNAL+ algorithm. Overall, Hap10 and Hap++ perform more or less on par, with a slight advantage for Hap10 at higher coverage levels, at lower molecule lengths and when more molecules receive the same barcode. This suggests the Hap10 assembly core is more robust to errors and problems due to imperfect 10X data. However, this comes at a cost: the Hap10 runtime is significantly higher. Table 1 reports CPU times for the results reported in Fig. 3. The pipelines were run on 24 CPU cores of a machine with 48 cores (Intel Xeon Silver 4116) and 754 GiB system memory. Clearly, the pre-processing by Hap++ occurs a time penalty, most visible for lower coverages, which pays off in a quicker runtime of the final SDhaP application,

Table 1 Run times (seconds) of the algorithms compared in Fig. 3

Ploidy	SNP rate	Coverage	SDhaP	Hap++	Hap10
Triploid	0.001	2	4	53	419
		5	24	54	2919
		10	34	92	1590
	0.01	2	3	64	2590
		5	150	162	8590
		10	660	400	19,221
Tetraploid	0.001	2	8	39	710
		5	21	90	4119
		10	65	204	13,824
	0.01	2	43	98	15,307
		5	478	370	28,598
		10	1497	1022	36,736

clearly seen at higher coverages. Hap10 is up to two orders of magnitude slower. When this is worth the effort, the pipeline can be run in *accurate mode* (using Hap10 optimization) with high haplotype quality, or in *fast mode* (using Hap++) with reasonable quality, depending on user preference.

Hap++ and Hap10 work on longer sequences

As a final test, we generated linked read data for the full chromosome 1 of the potato genome, simulating a tetraploid genome at a SNP rate of 0.01. The coverage is 10 per haploid genome. Results are reported in Table 2. Notably, SDhaP encountered a segmentation fault in this simulation, leaving us unable to report a result. Hap++ and Hap10 provide haplotypes with the same block lengths, with better accuracy in terms reconstruction rate and vector error rate. Moreover, the MEC between the read set and the reconstructed haplotypes is lower, suggesting a better compatibility between the two. However, as before, the computational cost of Hap10 is significant at approx. Nine hundred CPU hours vs. 12 h for Hap++. The results for H-PoP [9] on short reads show a very small haplotype length, but accurate, as expected.

Table 2 Results for chromosome 1 of a tetraploid potato with coverage 10 per haploid and a SNP rate of 0.01

Method	Avg. haplotype block length (no. SNPs)	N50 haplotype block length (bp)	Reconstruction rate	Vector error rate	MEC	CPU time (min)
SDhaP	–	–	–	–	–	–
H-PoP	12	1597	0.93	0.11	36,228	39
Hap++	3923	828,058	0.88	0.0083	342,956	741
Hap10	3923	828,058	0.92	0.0070	218,635	54,835

Real data

To obtain an idea of the applicability of Hap++ and Hap10 to real data, we ran the pipeline to reconstruct the six haplotypes of chromosome one of sweet potato (with the length of 36 Mb) based on 10X data available in the NCBI Short Read Archive.

The length distribution of the reconstructed haplotypes is displayed in Fig. 6; the N50 length of the blocks is 78.4 resp. 78.3 kb for Hap++ and Hap10. To compare the two plots, note that the SNP positions assigned to haplotype blocks are determined using the strongly connected components, which are the same for Hap++ and Hap10. Afterwards, the alternative optimization routine employed by Hap10 can yield different results than found by Hap++. The MEC scores between the read set and the reconstructed haplotypes

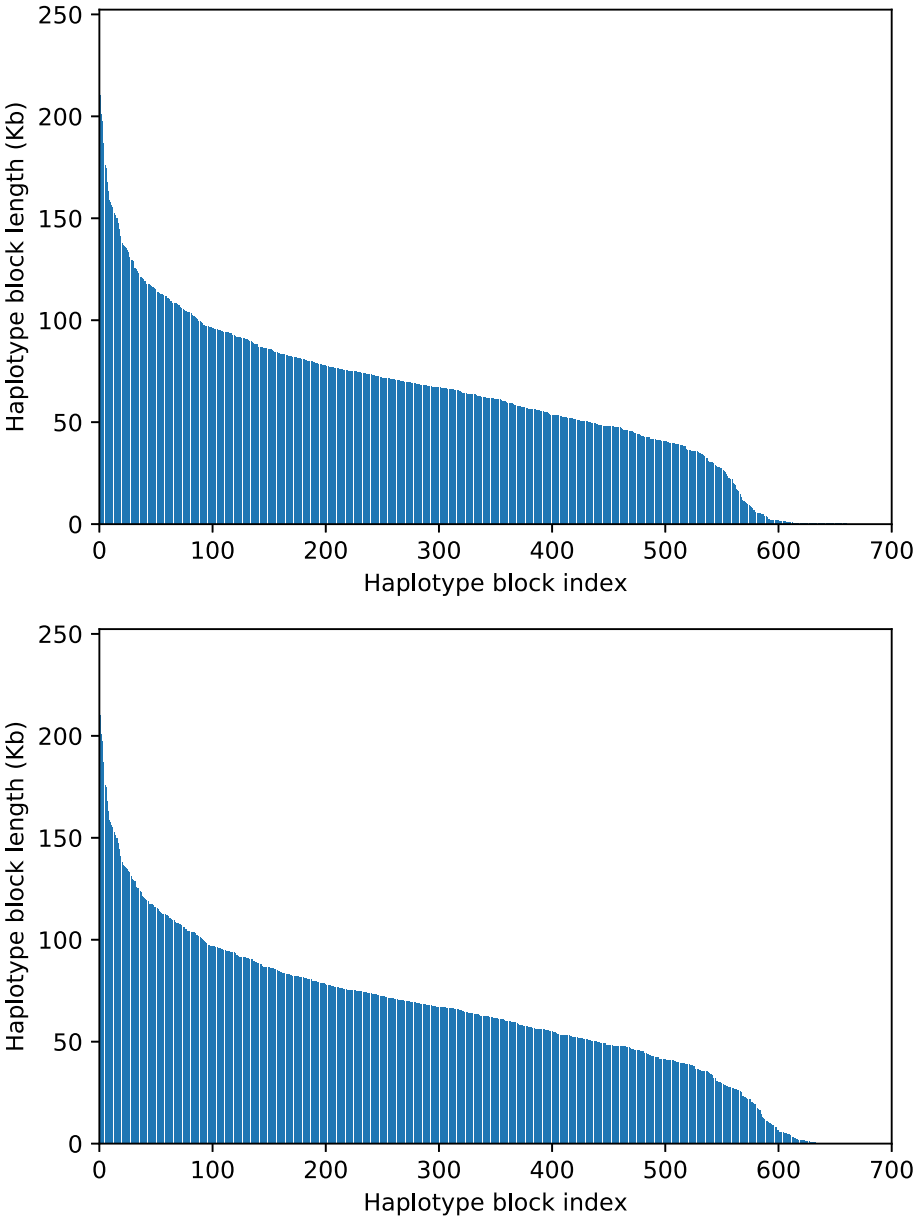

Fig. 6 Haplotype block length distributions for 10X real data of sweet potato using Hap++ (top) and Hap10 (bottom)

are 122,363 resp. 133,282 for the reconstructed haplotypes using Hap++ and Hap10, respectively, which would indicate that in this case the reconstructed haplotypes by Hap++ are more compatible with the read dataset than those generated by Hap10. However, true accuracy can only be evaluated by comparison to a ground truth.

Conclusion

We developed a first haplotyping pipeline specifically for linked-read data generated from a polyploid genome. It makes haplotyping full chromosomes of complex genomes feasible. The proposed Hap++ preprocessing pipeline improves on the accuracy of immediate application of SDhaP by approximately 30% (resp. 20%) on simulated 10X data of triploids (resp. tetraploids) at the cost of a decreased haplotype block length. Our framework builds on SDhaP, a typical Illumina haplotyping algorithm, using a standard fragment file as input. Any improvement in SDhaP or similar algorithms thus may immediately enhance linked-read (SLR) haplotyping. The proposed novel optimization scheme, Hap10, provides even more accurate haplotypes, albeit at significant computational cost.

One topic for future research is to consider different optimization techniques for the max-K-cut clustering problem [36]. A new method based on linear programming [37] may provide a solution for overcoming the high runtime involved in the semidefinite programming problem. A second avenue for research is automatic optimization of the key parameters of the pipeline, specifically the threshold t for the normalized cut algorithm and the convergence threshold used in the optimization step.

Supplementary information

Supplementary information accompanies this paper at https://doi.org/10.1186/s12859-020-03584-5.

Additional file 1: Figure S1. Description of the fragment file format using an example SAM and VCF input. **A)** In the SAM format, each line corresponds to a read (except for the header lines). The fourth column shows the genomic position of the first base of the aligned read. As a simple example, for each read 10 bases are shown in the 10th column. The next column shows the Phred quality of each base. Finally, the BX tag shows the barcode of each read provided by LongRanger software. For more information on the SAM format, see http://samtools.github.io. **B)** In the VCF format, the second column shows the genomic position of the variant in each line (except for the header lines). The third and fourth columns contain the reference and alternative alleles, respectively. The last column shows the genotype of the variant, in this example for a triploid. For more information on the SAM format see http://samtools.github.io. **C)** In fragment file designed for short reads, the first column shows the number of consecutive alleles (called part here) in the fragment, the second column the id of the fragment, the third column the start position of the first part, followed by the alleles of the part. The position is reported as the index of the variant in the VCF file, starting from 1. If there are more parts, they will appear next. The last column shows the Phred quality scores of all alleles in all parts consecutively. **D)** To include the barcode information for haplotyping, the barcodes in the SAM file BX tag are provided in the third column of the fragment file. The other columns are shifted accordingly. **E)** In Barcode-specific fragment file, reads with the same barcode are combined, as discussed in step one of "Hap++" Section. **F)** Molecule-specific fragments file is the output of step one of "Hap++" Section. The third column, which was the barcode, is iterated from one to the number of molecules for each barcode with an underscore in between. **G)** A schematic of the mentioned procedures is illustrated here. Eight fragments are presented in the image, colors indicating barcodes. In all three boxes, row corresponds to a line in the corresponding file.

Additional file 2: Figure S2. A graph indicating overlap between fragments. Red dots are vertices (corresponding to the fragments), grey lines are edges drawn when two fragments have at least one SNP in common. The depicted graph is for a case with 5 mb reference genome containing an N-region of 50 kb. The coverage is 15 per haploid and the SNP rate is 0.01. The average length of 10X DNA molecules for this simulation is set to 50 kb. Few fragments originate from a DNA molecule larger than 50 kb. The resulting graph has two separate subgraphs connected by a single edge. Note that one barcode-specific fragment connecting two read blocks is not sufficient for connecting the corresponding haplotypes. This phenomenon decreases the quality of reconstructed haplotype. The figure is generated using Cytoscape (www.cytoscape.org).

Additional file 3: Table S1. An example of the haplotype output format. We report the reconstructed haplotypes as a text file with a specific format similar to that of HapCUT2. Each haplotype block starts with a line describing the length of the haplotype, number of reads corresponding to the block and the minimum error

correction (MEC) score. From the next line, each row corresponds to each variant. The first and second columns show the 1-based index and variant position, respectively. Then, the next 2 ∗ ploidy columns are haplotypes and quality scores. For each allele of haplotypes, a quality score is provided. As a metric for quality, we use the number of matching reads at each position that are estimated for each haplotype.

Additional file 4: Table S2. The impact of the convergence threshold on Hap10 performance. A triploid genome of 230 kb with a SNP rate of 0.001 is simulated. The average molecule length and number of molecules per bead are 50 k and 10, respectively.

Additional file 5: Table S3. SDhaP with and without linked-read information. For the latter, the input data is considered as regular Illumina reads and barcodes are not used. The dataset is simulated using 1 Mb of chromosome one of potato genome with a SNP rate of 0.01. The coverage is 10. The results are averaged over 5 independent simulations. For the third row, we split the 1 Mb region into three independent parts of the same size. The last two rows present results of Hap++ and Hap10 on the same data.

Additional file 6: Table S4. Performance of SDhaP at different coverage levels, for a triploid genome with SNP rate of 0.001. The average molecule length and number of molecules per bead are 50 k and 10, respectively. The results are averaged over 5 independent simulations.

Abbreviations

BAM: Binary sequence alignment; bp: Base pair; extractHAIRs: Extract haplotype informative reads; MEC: Minimum error correction; MFR: Minimum fragment removal; NC: Normalized cut; RR: Reconstruction rate; SLRs: Synthetic long reads; SNP: Single nucleotide polymorphism; VCF: Variant call format

Acknowledgements

We would like to thank Brian Lavrijssen for helpful discussions.

Authors' contributions

Methods and experiments were designed by SM and DdR. Algorithm code was implemented by SM. SM and DdR wrote the manuscript. DdR and MHK supervised the project. All authors read and approved the final manuscript.

Funding

The authors received no specific funding for this work.

Availability of data and materials

- Reference genome of *Solanum tuberosum* ftp://ftp.ensemblgenomes.org/pub/plants/release-42/fasta/solanum_tuberosum/dna/
- 10X read data of sweet potato: https://www.ncbi.nlm.nih.gov/sra/SRX4706082
- LRSIM: https://github.com/aquaskyline/LRSIM
- LongRanger: https://github.com/10XGenomics/longranger
- FreeBayes: https://github.com/ekg/freebayes
- SDPNAL+: https://blog.nus.edu.sg/mattohkc/softwares/sdpnalplus/
- Scikit-learn: https://scikit-learn.org/
- SDhaP: https://sourceforge.net/projects/SDhaP/
- Our pipeline and code: https://github.com/smajidian/Hap10

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

Author Dick de Ridder is an Associate Editor for BMC Bioinformatics.

Author details

[1]School of Electrical Engineering, Iran University of Science & Technology, Narmak, Tehran 16846-13114, Iran. [2]Bioinformatics Group, Wageningen University, Droevendaalsesteeg 1, 6708PB, Wageningen, The Netherlands.

Received: 8 January 2020 Accepted: 5 June 2020
Published online: 18 June 2020

References

1. Comai L. The advantages and disadvantages of being polyploid. Nat Rev Genet. 2005;6(11):836–46.
2. Qian L, Hickey LT, Stahl A, Werner CR, Hayes B, Snowdon RJ, Voss-Fels KP. Exploring and harnessing haplotype diversity to improve yield stability in crops. Front Plant Sci. 2017;8:1534.
3. Liu PY, Zhang YY, Lu Y, Long JR, Shen H, Zhao LJ, et al. A survey of haplotype variants at several disease candidate genes: the importance of rare variants for complex diseases. J Med Genet. 2005;42(3):221–7.
4. Motazedi E, Finkers R, Maliepaard C, de Ridder D. Exploiting next-generation sequencing to solve the haplotyping puzzle in polyploids: a simulation study. Brief Bioinform. 2017;19(3):387–403.
5. Choi Y, Chan AP, Kirkness E, Telenti A, Schork NJ. Comparison of phasing strategies for whole human genomes. PLoS Genet. 2018;14(4):e1007308.

6. Zhang X, Wu R, Wang Y, Yu J, Tang H. Unzipping haplotypes in diploid and polyploid genomes. Comput Struct Biotechnol J. 2020;18:66–72.
7. Berger E, Yorukoglu D, Peng J, Berger B. HapTree: a novel Bayesian framework for single individual polyplotyping using NGS data. PLoS Comput Biol. 2014;10(3):e1003502.
8. Das S, Vikalo H. SDhaP: haplotype assembly for diploids and polyploids via semi-definite programming. BMC Genomics. 2015;16:260.
9. Xie M, Wu Q, Wang J, Jiang T. H-PoP and H-PoPG: heuristic partitioning algorithms for single individual haplotyping of polyploids. Bioinformatics. 2016;32(24):3735–44.
10. Siragusa E, Haiminen N, Finkers R, Visser R, Parida L. Haplotype assembly of autotetraploid potato using integer linear programming. Bioinformatics. 2019;35(21):4534.
11. Schrinner S, Mari RS, Ebler JW, Rautiainen M, Seillier L, Reimer J, Usadel B, Marschall T and Klau GW. "Haplotype threading: accurate polyploid phasing from long reads. 2020. BioRxiv. https://doi.org/10.1101/2020.02.04.933523..
12. He D, Saha S, Finkers R, Parida L. Efficient algorithms for polyploid haplotype phasing. BMC Genomics. 2018;19(Suppl 2): 171-80. Article number 110. https://doi.org/10.1186/s12864-018-4464-9.
13. Aguiar D, Istrail S. Haplotype assembly in polyploid genomes and identical by descent shared tracts. Bioinformatics. 2013;29(13):i352–60.
14. Moeinzadeh MH. De novo and haplotype assembly of polyploid genomes. PhD thesis. Germany: Freie Universität Berlin; 2019. http://dx.doi.org/10.17169/refubium-2712.
15. Goodwin S, McPherson JD, McCombie WR. Coming of age: ten years of next-generation sequencing technologies. Nat Rev Genet. 2016;17(6):333–51.
16. Wenger AM, Peluso P, Rowell WJ, Chang PC, Hall RJ, Concepcion GT, Ebler J, Fungtammasan A, Kolesnikov A, Olson ND, Töpfer A. Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome. Nat Biotechnol. 2019;37(10):1155–62.
17. Weisenfeld NI, Kumar V, Shah P, Church DM, Jaffe DB. Direct determination of diploid genome sequences. Genome Res. 2017;27(5):757–67.
18. Tolstoganov I, Bankevich A, Chen Z, Pevzner PA. cloudSPAdes: assembly of synthetic long reads using de Bruijn graphs. Bioinformatics. 2019;35.14:i61–70.
19. Marks P, Garcia S, Barrio AM, Belhocine K, Bernate J, Bharadwaj R, et al. Resolving the full spectrum of human genome variation using linked-reads. Genome Res. 2019;29(4):635–45.
20. Edge P, Bafna V, Bansal V. HapCUT2: robust and accurate haplotype assembly for diverse sequencing technologies. Genome Res. 2017;27(5):801–12.
21. Porubsky D, Garg S, Sanders AD, Korbel JO, Guryev V, Lansdorp PM, Marschall T. Dense and accurate whole-chromosome haplotyping of individual genomes. Nat Commun. 2017;8(1):1–10.
22. Majidian S, Kahaei MH. NGS based haplotype assembly using matrix completion. PLoS One. 2019;14(3):e0214455.
23. Garrison E, Marth G. Haplotype-based variant detection from short-read sequencing. 2012. arXiv preprint q-bio.GN/1207. 3907.
24. Motazedi E, de Ridder D, Finkers R, Baldwin S, Thomson S, Monaghan K, Maliepaard C. TriPoly: haplotype estimation for polyploids using sequencing data of related individuals. Bioinformatics. 2018;34(22):3864–72.
25. Comaniciu D, Meer P. Mean shift: a robust approach toward feature space analysis. IEEE Trans Pattern Anal Mach Intell. 2002;24(5):603–19.
26. Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: machine learning in Python. J Mach Learn Res. 2011;12:2825–30.
27. Shi J, Malik J. Normalized cuts and image segmentation. IEEE Trans Pattern Anal Mach Intell. 2000;22(8):888–905.
28. Frieze A, Jerrum M. Improved approximation algorithms for max *k*-cut and max bisection. Algorithmica. 1997;18(1):67–81.
29. de Klerk E, Pasechnik DV, Warners JP. On approximate graph colouring and max-*k*-cut algorithms based on the θ-function. J Comb Optim. 2004;8(3):267–94.
30. Boyd S. Vandenberghe L. Convex optimization: Cambridge University Press; 2004.
31. Rockafellar RT. Augmented Lagrangians and applications of the proximal point algorithm in convex programming. Math Oper Res USA. 1976;1(2):97–116.
32. Golub GH, Van Loan CF. Matrix computations: Johns Hopkins University Press; 1996.
33. Yang L, Sun D, Toh KC. SDPNAL++: a majorized semismooth Newton-CG augmented Lagrangian method for semidefinite programming with nonnegative constraints. Math Program Comput. 2015;7(3):331–66.
34. Luo R, Sedlazeck FJ, Darby CA, Kelly SM, Schatz MC. LRSim: a linked reads simulator generating insights for better genome partitioning. Comput Struct Biotechnol J. 2017;15:478–84.
35. Wu S, Lau KH, Cao Q, Hamilton JP, Sun H, Zhou C, et al. Genome sequences of two diploid wild relatives of cultivated sweetpotato reveal targets for genetic improvement. Nat Commun. 2018;9(1):4580.
36. Ghaddar B, Anjos MF, Liers F. A branch-and-cut algorithm based on semidefinite programming for the minimum *k*-partition problem. Ann Oper Res. 2011;188(1):155–74.
37. de Sousa VJR, Anjos MF, Le Digabel S. Improving the linear relaxation of maximum *k*-cut with semidefinite-based constraints. EURO J Comput Optimization. 2019;7(2):123–51.

Publisher's Note