Bioinformatics Advance Access published August 16, 2016

Genome analysis

H-PoP and H-PoPG: Heuristic Partitioning Algorithms for Single Individual Haplotyping of Polyploids

Minzhu Xie [1,*], Qiong Wu [2], Jianxin Wang [3] and Tao Jiang [4,5]

[1]Key Laboratory of Internet of Things Technologies and Application, College of Physics and Information Science, Hunan Normal University, Changsha 410081, China, [2]State Key Laboratory of Systematic and Evolutionary Botany, Institute of Botany, Chinese Academy of Sciences, Beijing 100093, China, [3]School of Information Science and Engineering, Central South University, Changsha 410083, China, [4]Department of Computer Science and Engineering, University of California, Riverside, CA 92521, USA and [5]MOE Key Lab of Bioinformatics and Bioinformatics Division, TNLIST / Department of Computer Science and Technology, Tsinghua University, Beijing, China

*To whom correspondence should be addressed.

Associate Editor: Dr. Inanc Birol

Abstract

Motivation: Some economically important plants including wheat and cotton have more than two copies of each chromosome. With the decreasing cost and increasing read length of next-generation sequencing technologies, reconstructing the multiple haplotypes of a polyploid genome from its sequence reads becomes practical. However, the computational challenge in polyploid haplotyping is much greater than that in diploid haplotyping, and there are few related methods.

Results: This paper models the polyploid haplotyping problem as an optimal poly-partition problem of the reads, called the Polyploid Balanced Optimal Partition (PBOP) model. For the reads sequenced from a k-ploid genome, the model tries to divide the reads into k groups such that the difference between the reads of the same group is minimized while the difference between the reads of different groups is maximized. When the genotype information is available, the model is extended to the Polyploid Balanced Optimal Partition with Genotype constraint (PBOPG) problem. These models are all NP-hard. We propose two heuristic algorithms, H-PoP and H-PoPG, based on dynamic programming and a strategy of limiting the number of intermediate solutions at each iteration, to solve the two models, respectively. Extensive experimental results on simulated and real data show that our algorithms can solve the models effectively, and are much faster and more accurate than the recent state-of-the-art polyploid haplotyping algorithms. The experiments also show that our algorithms can deal with long reads and deep read coverage effectively and accurately. Furthermore, H-PoP might be applied to help determine the ploidy of an organism.

Availability: https://github.com/MinzhuXie/H-PoPG

Contact: xieminzhu@hotmail.com

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

It is widely believed that domestication of wild plants was a key factor leading to human population expansion about 10 thousand years ago, and crops are still the most important source of human foods nowadays. Therefore, crop breeding is very important for the world food security. Most widely cultivated species of some economically important crops

*to whom correspondence should be addressed such as wheat, sugarcane, apple and banana are polyploids (Leitch and Leitch, 2008), *i.e.* they possess more than two copies of each chromosome. Polyploidy has long been regarded as an important feature to drive plant phenotypic diversification and hence, the investigation of polyploid genomic compositions will help understand plant evolution and crop improvement (Renny-Byfield and Wendel, 2014). With the recent rapid development of sequencing technologies, the read length of next-generation DNA sequencers has increased significantly and the sequencing cost has dropped enormously. Therefore, reconstructing the multiple haplotypes of a polyploid genome from its DNA reads is becoming a reality in practice.

There has been extensive research on the computational problem of reconstructing a pair of haplotypes from the DNA reads of a diploid genome, and many combinatorial optimization models (Xie *et al.*, 2010a; Browning and Browning, 2011) have been proposed, including MEC (minimum error correction) (Lippert *et al.*, 2002), MFR (minimum fragment removal), MSR (minimum SNP removal) (Lancia *et al.*, 2001), and two recent models MFC (maximum fragments cut) (Duitama *et al.*, 2010) and BOP (balanced optimal partition) (Xie *et al.*, 2012). Most of the above models and their extended versions are NP-hard (Bafna *et al.*, 2005; Cilibrasi *et al.*, 2007; Duitama *et al.*, 2010), and their exact algorithms run in time exponential in at least one input parameter (Bafna *et al.*, 2005; He *et al.*, 2010; Xie *et al.*, 2010b, 2008; Wang *et al.*, 2010; Bonizzoni *et al.*, 2015; Patterson *et al.*, 2015; Pirola *et al.*, 2015). Therefore, a large number of heuristic algorithms have been designed to deal with the problem (Panconesi and Sozio, 2004; Wang *et al.*, 2005; Genovese *et al.*, 2008; Duitama *et al.*, 2010; Xie *et al.*, 2012). In particular, integer linear programming has been adopted recently to solve MEC, but additional heuristic methods were also needed to process difficult blocks (Chen *et al.*, 2013). However, the computational complexity of polyploid haplotyping is much higher than that of diploid haplotyping. As shown in Figure 1, with two heterozygous bi-allelic loci, there are only two possible haplotype phasings for a diploid while there are eight possible haplotype phasings for a triploid. Generally speaking, with n heterozygous loci, there are $(k-1)^n$ different genotypes and at least $2^{n-1}(k-1)^n$ different haplotype phasings for a k-ploid. Simple extensions of current diploid haplotyping algorithms are usually inefficient in solving the polyploid haplotyping problem.

Recently, three polyploid haplotyping algorithms HapCompass (Aguiar and Istrail, 2013), HapTree (Berger *et al.*, 2014) and SDhaP (Das and Vikalo, 2015) have been introduced in the literature. The HapCompass algorithm converts a haplotype phasing as a spanning tree of the graph built from DNA reads, where the nodes represent single nucleotide polymorphisms (SNPs) and the edges indicate the evidence of co-occurring SNP alleles in a haplotype as supported by the reads. HapCompass tries to find a spanning tree with the minimum weighted edges removed from the graph based on cycle basis local optimization. HapTree uses a maximum-likelihood estimation framework and aims to find a haplotype phasing solution with the maximal likelihood explanation of the reads. To reduce computation complexity, HapTree adopts a strategy similar to dynamic programming, *i.e.* it finds a collection of high-likelihood phases of the first *n* SNP loci and then extends the phases to the next $n + 1$th SNP locus. SDhaP formulates the problem as a semi-definite program (SDP), and employs a low-rank Lagrangian scheme followed by randomized projections and a greedy refinement of the k-ploid haplotypes to solve the SDP (Das and Vikalo, 2015).

In this paper, we try to partition the DNA reads sequenced from a k-ploid organism into k groups such that the reads of the same group share the same alleles on as many SNP loci as possible and the reads from different groups are different on as many loci as possible. We balance both factors by proposing the Polyploid Balanced Optimal Partition (PBOP) model. Since genotype information is easy to obtain and the genotype information may help improve the accuracy of polyplotyping, we extend PBOP to PBOPG

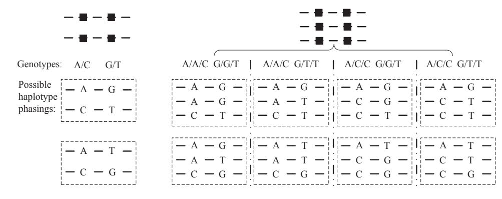


Fig. 1. The genotypes and corresponding haplotype phasings on two heterozygous loci.

(the Polyploid Balanced Optimal Partition with Genotype constraint) to make use of available genotype information. Both models are NP-hard. By limiting the number of intermediate solutions at each iteration of a dynamic programming, we designs two heuristic algorithms to solve the models. Extensive tests on simulated and real data show that our algorithms are much faster and more accurate than the recent state-of–the-art polyploid haplotyping algorithms, especially on data with long reads and deep read coverage.

2 Methods

2.1 Formulation and Problem

The input of the polyploid haplotyping problem consists of aligned DNA reads sequenced from a k-ploid organism. In our approach, we try to divide the reads into k different groups according to their original haplotypes. Since it is trivial to determine the alleles of the k-haplotypes at homozygous loci, and only loci where the reads have different alleles can be used to partition the reads into different groups, we will only keep alleles of the aligned reads on heterozygous SNP loci. The input aligned reads are denoted by an $m \times n$ SNP matrix M as in previous work (Xie *et al.*, 2008, 2012; Aguiar and Istrail, 2013; Berger *et al.*, 2014), where m is the number of reads and n the number of heterozygous SNP loci. $M[i, j]$, the entry of M at the ith row and jth column, encodes the allele of the ith read at the jth heterozygous SNP loci. $M[i, j]$ takes a value from $\{0, 1, -\}$, where '0' (or '1') represents the major allele (or the minor allele, respectively) at the locus in the population and '$-$' represents an unknown allele. The ith row (read) of M is denoted as r_i and the jth allele of r is denoted as $r[j]$. Similarly, a haplotype is denoted as a sequence of '0', '1' and '$-$', and the jth allele of a haplotype H is denoted as $H[j]$.

For two alleles $a_1, a_2 \in \{0, 1, -\}$, we define a similarity function $s(a_1, a_2)$ and dissimilarity function $d(a_1, a_2)$ as follows:

$$s(a_1, a_2) = \begin{cases} 1, & \text{if } a_1, a_2 \neq - \text{ and } a_1 = a_2; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

$$d(a_1, a_2) = \begin{cases} 1, & \text{if } a_1, a_2 \neq - \text{ and } a_1 \neq a_2; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Given a set $R = \{r_1, ..., r_p\}$ of p rows (reads), a k-partition function P of R is defined as a map $P : \{1, ..., p\} \to \{1, ..., k\}$, which means that r_i is put into the $P(i)$th group (subset) $G_{P(i)}$ according to the function P. Since every row in R has a unique index, the row with the smallest index in a group is called the *representative* row of the group, and the smallest index of the rows in a group G is denoted by $I(G)$. If $G = \varnothing$, let $I(G) = p + 1$. Let the groups obtained by applying P on R be $G_1^P, ..., G_k^P$. A k-partition function P is called a *canonical* k-partition function if the following condition holds: for any two groups G_i^P and G_j^P with $1 \le i < j \le k$, $I(G_i^P) < I(G_j^P)$.

The consensus haplotype H_i of the ith group G_i is defined as the haplotype that maximizes the sum of similarities between the reads in the group G_i and the haplotype, $i.e.$

$$H_i = \underset{H}{\operatorname{argmax}} \sum_{r \in G_i} \sum_{j=1}^{n} s(r[j], H[j]). \tag{3}$$

Assume that the consensus haplotypes are the original haplotypes, and the difference between the consensus haplotypes and the reads can be regarded as sequencing errors. Given a k-partition function P on a set of reads (rows), the corrected error measure $C(P)$ is defined as the total difference between the reads and the consensus haplotypes of their groups, $i.e.$

$$C(P) = \sum_{i=1}^{k} \sum_{r \in G_i} \sum_{j=1}^{n} d(r[j], H_i[j]). \tag{4}$$

Since we only consider heterozygous loci, any two of the k haplotypes should not be identical when several consecutive heterozygous loci are considered together, and it is desirable that the k consensus haplotypes are distinctively different. We introduce another measure $D(P)$ as follows, which is called the diversity measure:

$$D(P) = \sum_{i_1, i_2 = 1,\dots,k; i_1 \neq i_2} \left(\sum_{j=1}^{n} d(H_{i_1}[j], H_{i_2}[j]) \right). \tag{5}$$

We would like to find an ideal partition P that makes the difference between the reads of a same group minimized and the difference between the reads of different groups maximized, $i.e.$ one that minimizes $C(P)$ and maximizes $D(P)$. However, this may be impossible in most cases, since a partition function P minimizing $C(P)$ may not ensure that $D(P)$ is maximized and vice versa. Therefore, we combine both measures into a weighted partition score:

$$s(P) = (1 - w)D(P) - wC(P), \tag{6}$$

where w is a weighting parameter with $0 \leq w \leq 1$. In the following, we propose a new optimization model for the polyploid haplotyping problem.
Polyploid Balanced Optimal Partition (PBOP): Given an SNP matrix M and weight w, find a k-partition function P of the rows in M such that $s(P) = (1 - w)D(P) - wC(P)$ is maximized.

Note that when $w = 1$ and $k = 2$, PBOP becomes the MEC model for the diploid haplotype assembly problem. MEC is known to be NP-hard (Lippert *et al.*, 2002). In fact, it has been recently shown not to be in APX under the Unique Games Conjecture (Bonizzoni *et al.*, 2015). Therefore, PBOP is also NP-hard, and not in APX under the Unique Games Conjecture. In the next subsection, we introduce a heuristic algorithm called *H-PoP*.

When the genotype $\mathcal{G}$ of the polyploid is known, we will also consider the genotype constrained version of PBOP. Since we do not consider homozygous loci, for each locus j, $\mathcal{G}[j] \in \{1, ..., k-1\}$. $\mathcal{G}[j] = t$ means that at the jth locus, there are t haplotypes taking the allele '1' and the other $k - t$ haplotypes taking the allele '0'.

Let $\mathcal{I}_i(x)$ be an indicator function, *i.e.* $\mathcal{I}_i(x) = 1$ if $x = i$; otherwise 0. Given a k-partition function P on a set of rows $R = \{r_1, ..., r_p\}$, the consensus haplotypes $H'_1, ..., H'_k$ with genotype constraint $\mathcal{G}$ are the haplotypes that satisfy the following conditions: (i) $H'_i[j] = -$ when there are no reads in group G_i covering the jth locus (*i.e.* the alleles at locus j of the reads in group G_i are all unknown), (ii) $\sum_{i=1}^{k} \mathcal{I}_1(H'_i[j]) \leq \mathcal{G}[j], \sum_{i=1}^{k} \mathcal{I}_0(H'_i[j]) \leq k - \mathcal{G}[j]$ for each locus $j \in \{1, ..., n\}$ and (iii) $\sum_{i=1}^{k} \sum_{r \in G_i} \sum_{j=1}^{n} s(r[j], H'_i[j])$ is maximized.

The genotype constrained corrected error measure $C'(P)$ and partition score $s'(P)$ are defined as:

$$C'(P) = \sum_{i=1}^{k} \sum_{r \in G_i} \sum_{j=1}^{n} d(r[j], H'_i[j]). \tag{7}$$

Similarly, the genotype constrained diversity measure $D'(P)$ and partition score $s'(P)$ are defined as follows:

$$D'(P) = \sum_{i_1, i_2 = 1,\dots,k; i_1 \neq i_2} \left(\sum_{j=1}^{n} d(H'_{i_1}[j], H'_{i_2}[j]) \right); \tag{8}$$

$$s'(P) = (1 - w)D'(P) - wC'(P). \tag{9}$$

Polyploid Balanced Optimal Partition with Genotype constraint (PBOPG): Given an SNP matrix M, the genotype $\mathcal{G}$ of the polyploid and weight w, find a k-partition function P of the rows in M such that $s'(P) = (1 - w)D'(P) - wC'(P)$ is maximized.

2.2 Algorithm

Given an $m \times n$ SNP matrix M, the number of ways to partition m different rows $r_1, ..., r_m$ into k non-empty groups is a Stirling number of the second kind, which is denoted as $S(m, k)$. Since $S(m, k) = \frac{1}{k!} \sum_{i=0}^{k} (-1)^i \binom{k}{i} (k - i)^m$, when m is large, it is impractical to enumerate all possible partitions and choose one with the maximum partition score. To solve the PBOP model of M efficiently, we propose a heuristic dynamic programming algorithm in the subsection. In other words, we will consider solutions for a number of rows of M then extend the solutions to the next row, and so on until all rows of M have been considered.

We first introduce some definitions and notations similar to those in (Xie *et al.*, 2012), but some of which have different meanings. Let $b(i)$ ($e(i)$) denote the first (the last) column at which the ith row of M takes non-'$-$' values. If and only if $b(i) \leq j \leq e(i)$, row i *spans* column j. $R(j)$ denotes the set of rows that contain the rows in M spanning the jth column.

In the following, we assume that M has been sorted such that for two rows i_1 and i_2 of M with $i_1 < i_2$, $b(i_1) < b(i_2)$, or $b(i_1) = b(i_2)$ and $e(i_1) \leq e(i_2)$.

Let P be a canonical k-partition function on a set of rows R and P' a canonical k-partition function on a subset R' of R. If for any two rows $i, j \in R'$, $P'(i) = P'(j)$ if and only if $P(i) = P(j)$, P' is called the *projection* of P on R' and P an *extension* of P' on R. It is easy to verify that given a partition P of R and a subset R', the projection of P on R' is unique, but not vice verse. The projection of P on R' is denoted by $P[R']$ for convenience.

Let P be a canonical k-partition function of the subset $R = \{ r_{i_1}, ..., r_{i_q} \}$ of the rows of M with i_q as the largest row index in R, and $R' = \{r_1, ..., r_{i_q}\}$ ($i.e.$ the set of all rows from the first row to row r_{i_q}). P' is an *optimal extension* of P if the following conditions hold: (i) P' is an extension of P on R' and (ii) for any possible extension P'' of P on R', $s(P') \geq s(P'')$.

Similarly, when the genotype $\mathcal{G}$ is available, P' is a genotype constrained *optimal extension* of P if the following conditions hold: (i) P' is an extension of P on R' and (ii) for any possible extension P'' of P on R', $s'(P') \geq s'(P'')$.

Given a partition P of R, let $E(P)$ denote an optimal extension of P and $E'(P)$ a genotype constrained optimal extension of P. We call $s(E(P))$ (or $s'(E'(P))$) the global (or the genotype constrained global)

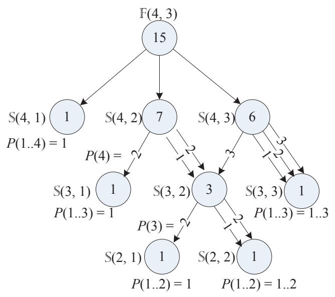


Fig. 2. An illustration of $\mathbb{F}(4,3)$, the set of all 15 canonical functions that partition 4 rows into at most 3 groups. A partition function is represented by a path from the root node to a leaf node. For example, the path from the root node $\mathbb{F}(4,3)$ to the leaf node $\mathbb{S}(3,1)$ represents such a function P with $P(4) = 2$ and $P(i) = 1$ for $i = 1$ to 3.

score of P and denote it as $s_e(P)$ (or $s'_e(P)$, respectively) for convenience. The following theorem is straightforward.

Theorem 1. *Let P be a canonical k-partition function of $R(n)$ for an $m \times n$ SNP matrix M. $E(P)$ is a solution to PBOP (or PBOPG) of M if and only if the following condition holds: for any possible canonical k-partition function P' of $R(n)$, $s_e(P) \geq s_e(P')$ (or $s'_e(P) \geq s'_e(P')$, respectively).*

We consider the set of canonical k-partition functions that partition a set of i rows into at most k groups, and denote the function set as $\mathbb{F}(i,k)$. Let $\mathbb{S}(p,q)$ denote the set of canonical functions that partition p rows into q non-empty groups. It is obvious that $\mathbb{F}(i,k) = \cup_{q=1..t}\mathbb{S}(i,q)$, where t is the maximum of i and k.

Let p and q be positive integers. To enumerate all canonical functions that partition p rows into q non-empty groups, we consider the following cases:

- $p < q$: Such functions do not exist, *i.e.* $\mathbb{S}(p,q) = \varnothing$.
- $p = q$:It is easy to verify that $\mathbb{S}(p,q)$ contains only one partition function, *i.e.* $P(i) = i$ for $i = 1$ to p (denoted by $P(1..p) = 1..p$).
- $p > q$: If $q = 1$, there is only one partition function in $\mathbb{S}(p,q)$, *i.e.* $P(i) = 1$ for $i = 1$ to p (denoted by $P(1..p) = 1$). If $q > 1$, we may construct $\mathbb{S}(p,q)$ using the following recursive process. Considering the last row p, the group containing row p either has only one row, *i.e.* row p, or two or more rows. We denote the set of partition functions that put row p into a group by itself (labeled as group q) and partition the first $p-1$ rows into the other $q-1$ non-empty groups as $\{(P(p) = q) \cdot \mathbb{S}(p-1, q-1)\}$, and denote the the set of partition functions that partition the first $p-1$ rows into q non-empty groups and put row p into one of the q groups as $\{(P(p) = 1, ..., q) \cdot S(p-1, q)\}$. Then, we have the following recurrence:

$$\begin{aligned}\mathbb{S}(p,q) &= \{(P(p) = q) \cdot \mathbb{S}(p-1, q-1)\} \cup \\ &\quad \{(P(p) = 1, ..., q) \cdot \mathbb{S}(p-1, q)\}.\end{aligned}$$

Figure 2 illustrates $\mathbb{F}(4,3)$, the set of all 15 different functions partitioning 4 rows into at most 3 groups. In the graph, the values $\mathbb{S}(p,q)$ with $p = q$ or $q = 1$ are represented as the leaf nodes and a partition function is represented by a path from the root node to a leaf node. Please note that there may be more than one path from the root to a leaf node. For example, there are three different paths from the root $\mathbb{F}(4,3)$ to the node $\mathbb{S}(3,3)$, corresponding to three different functions: (i) $P(4) = 1, P(1..3) = 1..3$, (ii) $P(4) = 2, P(1..3) = 1..3$ and (iii) $P(4) = 3, P(1..3) = 1..3$.

Given an $m \times n$ SNP matrix M, our algorithm considers the set $R(1)$ of rows spanning column 1 first. To make the algorithm scalable, we will be able to enumerate all the partition functions on a small number of rows. Let q be the maximum integer such that $\mid \mathbb{F}(q,k) \mid \leq 1000$ and $q \leq \mid R(1) \mid$. Let R_f be the set of the first q rows in $R(1)$ and R_b contain the rest of the rows in $R(1)$. When $R(1)$ contains no more than q rows, $R_f = R(1)$ and $R_b = \varnothing$. Since the number of rows in R_f is at most q and $\mid \mathbb{F}(q,k) \mid \leq 1000$, it is practical to enumerate all partition functions of R_f by the above method.

For each group G obtained by partitioning R_f according to some function P, let $b(G) = \min_{r \in G} b(r)$ and $e(G) = \max_{r \in G} e(r)$. We use a $2 \times n$ matrix to record a profile $\mathbb{P}(G)$ for G, where the element $\mathbb{P}(G)[v,j]$ counts the number of rows of G whose values at column j is v for $v = 0, 1$ and $j = b(G), ..., e(G)$. The set of the profiles for all groups incurred by P is denoted as $\mathbb{P}_P$.

Based on $\mathbb{P}(G)$, we obtain a consensus haplotype H_G of G easily:

$$H_G[j] = \begin{cases} 0, & \text{if } \mathbb{P}(G)[0,j] > \mathbb{P}(G)[1,j] \\ 1, & \text{if } \mathbb{P}(G)[0,j] < \mathbb{P}(G)[1,j] \\ -, & \text{otherwise} \end{cases} \quad \text{for } j = b(G), ..., e(G). \tag{10}$$

Using equations (4) - (6) and (10), the score $s(P)$ of a $k-$partition function P on R_f can be calculated in time $O(l_m(q + k^2))$, where $l_m = \max_{i \in R_f} e(i) - \min_{i \in R_f} b(i) + 1$.

Recall that at the very beginning of the algorithm, $R_f = \{r_1, ..., r_q\}$, and both the optimal extension and the genotype constrained optimal extension of a partition function P of R_f are P itself, *i.e.* $E(P) = P$ and $E'(P) = P$. Therefore, $s_e(P) = s(P)$ and $s'_e(P) = s'(P)$.

For an optimal partition function P of the set of the rows in M, the projection of P on R_f is likely a suboptimal partition function of R_f. In the following extension from R_f to rows in R_b, we will only consider the top $10k^2$ partition functions of R_f, which is denoted by F, based on empirical experience.

If R_b is not empty, let r be the row with the smallest index in R_b and $R'_f = R_f \cup \{r\}$. We construct all possible extensions of the functions in F to R'_f and record (at most) $10k^2$ extensions with the highest global scores in F'.

For a partition function P on R_f, there are at most k extensions of P on R'_f. Let the number of rows in R_f be q_f and $g = \max_{i=1..q_f} P(i)$, *i.e.* P partitions the q_f rows of R_f into g nonempty groups $G_1, ..., G_g$. Since the row r can be put into any one of the g groups, there are at least g extensions $P'_1, ..., P'_g$ of P on R'_f. When $g < k$, r can be put into a new group and hence there is an additional extension P'_{g+1} of P. For each extension P'_t,

$$P'_t(i) = \begin{cases} P(i), & 1 \leq i \leq q_f; \\ t, & i = q_f + 1. \end{cases} \tag{11}$$

Let the nonempty groups obtained by applying P'_t on R'_f be $G'_1, ..., G'_{g'}$. Since the difference Δs between the global scores of P'_t and P is due to putting r in G'_t, it is easy to compute by considering two cases below.

- $t \leq g$: In this case, $g' = g$. For $i = 1, ..., g$ except $i = t$, $G'_i = G_i$, $\mathbb{P}(G'_i) = \mathbb{P}(G_i)$ and $H_{G'_i} = H_{G_i}$. Moreover, $G'_t = G_t \cup \{r\}$, $\mathbb{P}(G'_t)[v,j] = \mathbb{P}(G_t)[v,j] + I_r[v,j]$ for $v = 0, 1$ and $j = b(G'_t), ..., e(G'_t)$, where $I_r[v,j]$ = 1 when r takes value v at column j; otherwise $I_r[v,j]$ = 0. Use Equation (10) to compute $H_{G'_t}$. Let the set of columns where $H_{G_t} \neq H_{G'_t}$ be $\mathbb{C}$. Then

$$\begin{aligned}\Delta C &= \sum_{j \in \mathbb{C}} \mid \mathbb{P}(G_t)[0,j] - \mathbb{P}(G_t)[1,j] \mid + \sum_{j=b(r)}^{e(r)} d(H_{G'_t}[j], r[j]) \\ \Delta D &= \sum_{j \in \mathbb{C}} \sum_{i \neq t} \Big(d(H_{G'_t}[j], H_{G_i}[j]) - d(H_{G_t}[j], H_{G_i}[j]) \Big) \\ \Delta s &= (1-w)\Delta D - w\Delta C \end{aligned} \tag{12}$$

- $t > g$: In this case, $g < k$ and $g' = t = g + 1$. For $i = 1, ..., g$, $G'_i = G_i, \mathbb{P}(G'_i) = \mathbb{P}(G_i)$ and $H_{G'_i} = H_{G_i}$. Moreover, $G'_t = \{r\}$, $\mathbb{P}(G'_t)[v, j] = I_r[v, j]$ for $v = 0, 1$ and $j = b(r), ..., e(r)$, where $I_r[v, j]$ = 1 when r takes value v at column j; otherwise $I_r[v, j]$ = 0. $H_{G'_t}$ can be obtained easily using Equation (10). Then

$$\Delta s = (1-w)\Delta D = (1-w)\sum_{j=b(r)}^{e(r)}\sum_{i=1}^{g}\Big(d(H_{G'_t}[j], H_{G_i}[j])\Big) \tag{13}$$

The global score of P'_t is

$$s_e(P'_t) = s_e(P) + \Delta s, \tag{14}$$

which can be calculated in time $O(lk)$, where $l = e(r) - b(r) + 1$.

Once all possible extensions of the functions in F to R'_f have been enumerated, we set $F = F'$, $R_f = R_f \cup \{r\}$ and $R_b = R_b - \{r\}$. The above process is repeated until R_b becomes empty and $R_f = R(1)$.

After obtaining the top $10k^2$ partition functions of $R(j)$ for some j, which are again stored in F, we consider the next column $j+1$. Let $R_d = R(j) - R(j+1)$, $R_r = R(j) \cap R(j+1)$ and $R_b = R(j+1) - R(j)$. We first compute the projections of the functions in F on R_r, and then extend them to $R(j+1)$.

Let F' be the set of the projections of all functions in F on R_r. It is obvious that $\mid F' \mid \leq \mid F \mid$. For a function P in F, we can easily compute its projection P' on R_r. If $R_d = \varnothing$, *i.e.* $R_r = R(j)$, then $P' = P$ and $F' = F$; otherwise, we compute its projection P' as follows. Let the rows in R_r be $r_{i_1}, r_{i_2}, ...,$ and r_{i_z}, and P'' be such a partition function of R_r that $P''(r_i) = P(r_i)$ for $i = i_1, ..., i_z$. Suppose that P'' partitions R_r into t non-empty groups $G''_1, ..., G''_t$. Sort these groups by their representative row indices in the ascending order and let rank(g) be the rank of G''_g in the sorted groups, *i.e.* if the representative row index of G''_g is the p smallest then rank$(g) = p$. Set $P'(r_i) = \text{rank}(P''(r_i))$ for $i = i_1, ..., i_z$, and then P' is the projection of P on R_r.

For each function P' in F', we can calculate its global score $s_e(P')$ using the equation below:

$$s_e(P') = \max_{P\in F \text{ and } P' \text{ is a projection of } P} s_e(P). \tag{15}$$

The time complexity of computing F' and the scores is $O(|F|(k\log k + |R(j)|))$. Once F' and the corresponding scores have been computed, set $F = F'$ and $R_f = R_r$, and extend the functions in F to $R(j+1)$ by deleting a row from R_b and adding it to R_f, one at a time, until $R_f = R(j+1)$ as done above by using Equation (11).

The above iteration continues until $R(j)$ contains the last row of M. Finally, an extension of the function in F with the highest global score is output as a solution to the PBOP problem of M. A pseudo code for this algorithm, called H-PoP, is presented at Figure S1 in the Supplementary Materials.

For the PBOPG model, a similar heuristic algorithm H-PoPG can be easily devised. All we need is a simple modification of H-PoP to make use of the provided genotype information. Please see the Supplementary Materials for the details of H-PoPG.

3 Results

We use both real data and simulated data to compare the performance of our algorithms H-PoP, H-PoPG and three recent single individual polyplotyping algorithms HapCompass (Aguiar and Istrail, 2013), HapTree (Berger *et al.*, 2014) and SDhaP (Das and Vikalo, 2015). Besides aligned SNP reads, all algorithms except H-PoP and SDhaP require genotype information as an additional input. The weight w is set as 0.9 for H-PoP and H-PoPG unless otherwise specified. All tests are conducted on some 64 bit nodes with 2.6GHz CPU and 128GB RAM of a Linux cluster, and each result on simulated data is the average of 100 repeated tests with the same parameters.

3.1 Results on Real Data

Using 454 GS FLX Titanium sequencing, Curtin et al. (Curtin *et al.*, 2012) obtained a 12.7 Mb assembly of AWRI1499, a prevalent wine spoilage strain of the yeast species Dekkera bruxellensis. The assembly is comprised of 324 contigs (N50 = 68 kb) in 99 scaffolds, at median read coverage of 26-fold and it was found that AWRI1499 is a triploid (Curtin *et al.*, 2012). We downloaded the read data from the SRA database of NCBI under the accessions SRX327045 and SRX327033, and the 14 contigs of the first scaffold from the Assembly database of NCBI under the accession AHIQ01. After cleaning adapters and low quality bases from the reads, BWA-MEM (Li and Durbin, 2009) was used to align the reads against the contigs with default parameters, and the SAMtools package (Li *et al.*, 2009; Li, 2011a,b) was used to call SNPs based on the aligned reads. The genotype of an SNP locus is determined by the proportion of the alleles 0 and 1 on the locus. Keeping only alleles at the heterozygous SNP loci of the aligned reads, we obtained 14 SNP matrices, one for each contig.

We tested H-PoPG, H-PoP, HapTree, HapCompass and SDhaP on these real data. AWRI1499 is a triploid, but its three true haplotypes are unavailable. Instead, there are only consensus contigs. Since the true haplotypes are unknown, we use the MEC score (Lippert *et al.*, 2002) to evaluate the accuracy of the reconstructed haplotypes. More precisely, given an $m \times n$ SNP matrix M, let the reconstructed k haplotypes be $\mathcal{H}$ = (H_1,..., H_k). The MEC score $s_c(\mathcal{H}, M)$ is the minimum number of sequencing errors in the SNP matrix if $\mathcal{H}$ is considered as the true haplotypes. That is,

$$s_c(\mathcal{H}, M) = \sum_{i=1,...,m} \min_{p=1,...,k} \big(\sum_{j=1,...,n} d(M[i,j], H_p[j]) \big), \tag{16}$$

where $d(.,.)$ is the dissimilarity function given in Equation 2. The MEC rate $e_c(\mathcal{H}, M)$ is the minimum sequencing error rate of the corresponding DNA reads at the SNP loci if $\mathcal{H}$ is considered as the true haplotypes, *i.e.* $e_c(\mathcal{H}, M) = s_c(\mathcal{H}, M)/$ the number of non- '-' elements of M.

The detailed test results on all 14 individual contigs are given in Tables S1-S14 of the Supplementary Materials. H-PoPG, H-PoP, HapCompass and SDhaP were able to reconstruct the haplotypes from the SNP matrices for all contigs, but the performance of HapCompass was clearly inferior to the other algorithms. H-PoPG and HapTree achieved similar MEC rates. Without the genotype constraint, SDhaP had less MEC rates while H-PoP obtained the least MEC rates. However, HapTree aborted with run-time errors on the data of contigs 6, 8, 15, 17, and 21, and failed to terminate in seven days on the data of contig 19. The SNP matrices of contigs 6, 8, 15, 17, 19 and 21 consist of 64223 rows and 12226 columns (*i.e.* SNPs). The total number of non-'-' elements is 448619, the average read coverage of each SNP is 36.7, and the average number of non-'-' elements in a row (*i.e.* an SNP read) is 7.0. All algorithms except HapTree reconstructed 20 disjoint blocks of haplotypes on these contigs, with the average block length being 611.3 SNPs. The test results on the 6 contigs are summarized in Table 1.

The SNP matrices of the remaining 8 contigs (contigs 1, 2, 4, 7, 10, 12, 16 and 18) consist of 13943 rows and 4096 columns (SNPs). The total number of non-'-' elements is 115371, the average read coverage of each SNP is 28.2, and the average number of non-'-' elements in a row (an SNP read) is 8.3. All algorithms reconstructed 98 disjoint blocks of haplotypes on these contigs, with the average block length being 41.8 SNPs. The test results on the 8 contigs are presented in Table 2.

Table 1. Comparison of the performance of H-PoPG, H-PoP, HapCompass (HapC in the table) and SDhaP on the real data corresponding to contigs 6, 8, 15, 17, 19 and 21 of triploid AWRI1499.

	$k = 3$				$k = 2$
	H-PoPG	H-PoP	HapC	SDhaP	H-PoP
MEC	8867	3891	22822	8186	28400
MEC rate (%)	1.98	0.87	5.08	1.82	6.33
Phased SNPs	12223	12160	12226	12226	12210
Time (s)	19.1	13.9	25237.9	41425.9	7.5
Memory (GB)	8.2	8.2	43.5	32.4	4.3

Table 2 shows that when regarding AWRI1499 as a diploid (*i.e.* set $k = 2$), the MEC rate of H-PoP was 4.33%, which is much larger than the raw single base sequencing error rate of 2.23% shown in the Supplementary Figure 8 of Margulies *et al.* (2005). This suggests that we could perhaps use H-PoP to help determine the ploidy of an organism by trying different k and comparing the MEC rate and DNA sequencing error rate. When regarding AWRI1499 a triploid, without applying the genotype constraint, the MEC rate 0.43% achieved by H-PoP was the least, about a half of that of SDhaP. With the genotype constraint, the MEC rate 1.28 of H-PoPG was a little better than that of HapTree. The MEC rate 3.25 of HapCompass was the largest, which suggests that the reconstructed haplotypes of HapCompass may be inaccurate. The phased SNPs of HapTree, HapCompass and SDhaP are 4096, while two of these SNPs were not phased by H-PoPG since it does not perform phasing at loci that are not covered by reads in the corresponding groups. As for running time, H-PoP and H-PoPG were more than 40 times faster than HapTree, HapCompass and SDhaP. The maximum run-time (resident) memory used by HapCompass and SDhap was more than 20GB, while the other algorithms needed only less than 3GB memory.

3.2 Results on Simulated Data

We generated aligned SNP reads of a polyploid genome as follows. First, k genomes were generated based on the real contigs and VCF files of AWRI1499. Given a contig as the haplotype template and a corresponding VCF file, we generated k copies of the contig as the initial k genomes. For each heterozygous SNP in the VCF file, a genotype g (the number of alternative alleles) of the SNP was generated randomly following a uniform distribution from 1 to $k - 1$. Then g genomes were randomly selected from the k genomes and their alleles at the SNP locus were set as the alternative alleles, while the other genomes were set as the reference alleles. The generated k genomes were saved in a FASTA format file named k-ploid.fa. Second, we used ART (Huang *et al.*, 2012), a next-generation sequencing read simulator, to generate simulated reads from the the k genomes.

To simulate the real data of AWRI1499, we ran ART with the 454 GS FLX Titanium platform profile that came with the simulator to generate single-end reads and paired-end reads. ART requires a parameter, the coverage c of each haplotype, to generate single-end reads, and three parameters, the coverage c of each haplotype, the mean insert length f and the standard deviation σ of insert length, to generate paired-end reads. In the following tests without explicit specification, f was set as 800 and σ was set as 150 according to the distribution of the reads in the real data. The single-end and paired-end reads in the same dataset were generated by ART with the same coverage parameter.

Correct phasing rate (R_c), and *perfect solution rate* (R_p) will be used to measure the phasing accuracy of the k haplotypes reconstructed by an algorithm. When the genotype of the reconstructed haplotypes equals to the original genotype, another measure *vector error rate* (R_v) (Berger *et al.*, 2014) is used too. Let the set of k haplotypes reconstructed by an algorithm be $\mathbf{H} = \{H_1, H_2, ..., H_k\}$ and the set of true haplotypes be

Table 2. Comparison of the performance of H-PoPG (H-PG in the table), H-PoP, HapTree (HapT in the table), HapCompass (HapC in the table) and SDhaP on the real data corresponding to contigs 1, 2, 4, 7, 10, 12, 16 and 18 of triploid AWRI1499.

	$k = 3$					$k = 2$
	H-PG	H-PoP	HapT	HapC	SDhaP	H-PoP
MEC	1471	497	1487	3748	1085	4991
MEC rate (%)	1.28	0.43	1.29	3.25	0.94	4.33
Phased SNPs	4094	4061	4096	4096	4096	4082
Time (s)	6.5	4.4	371.5	575.8	345.3	4.4
Memory (GB)	2.2	2.2	1.3	39.1	25.6	0.8

$\mathbf{H}^* = \{H_1^*, H_2^*, ..., H_k^*\}$. Let $\mathcal{H}$ be a one-to-one mapping from $\mathbf{H}$ to $\mathbf{H}^*$. Define $M(\mathcal{H}) = \sum_{i=1}^{k} \sum_{j=1}^{n} s(H_i[j], \mathcal{H}(H_i)[j])$, where s is the similarity function given in Equation (1). The correct phasing rate R_c is defined as follows:

$$R_c(\mathbf{H}) = \max_{\mathcal{H}} M(\mathcal{H})/nk,$$

where n is the number of phased SNPs. For the example, in Figure 3, the correct phasing rate $R_c(\mathbf{H}) = 10/12$. The perfect solution rate is $R_p = n_c/k$, where n_c is the number of haplotypes correctly reconstructed ($i.e.$ they are exactly the same as the true ones) by the algorithm. For example, the perfect solution rate of $\mathbf{H}$ in Figure 3 is 1/3.

The vector error measure is a generalization of the switch error measure for diploid haplotype assembly to polyploid phasing. The vector errors in k reconstructed haplotypes are defined in (Berger *et al.*, 2014) as the minimum number of segments on all chromosomes for which a switch must occur. To make it easy to understand, we give another equivalent definition of the measure here. A one-to-one mapping $\mathcal{H}$ from $\mathbf{H}$ to $\mathbf{H}^*$ is called a matching at locus j if $\sum_{i=1}^{k} d(H_i[j], \mathcal{H}(H_i)[j]) = 0$. The distance $d(\mathcal{H}_1, \mathcal{H}_2)$ between two mappings $\mathcal{H}_1$ and $\mathcal{H}_2$ is defined as $\sum_{i=1}^{k} I(\mathcal{H}_1(H_i) \neq \mathcal{H}_2(H_i))$, where I is an indicator function (*i.e.* $I(a)$ = 1 if a is a true statement and $I(a)$ = 0 otherwise). When the genotypes of $\mathbf{H}$ and $\mathbf{H}^*$ are equal, there exist a series of one-to-one mappings $\mathcal{M} = \{\mathcal{H}_1, ..., \mathcal{H}_n\}$ such that $\mathcal{H}_j$ is a matching at locus j for $j = 1, ..., n$, and such a sequence is called a matching sequence. For a matching sequence $\mathcal{M}$, the total number of changes between adjacent matchings is $T_c(\mathcal{M}) = \sum_{j=1}^{n-1} d(\mathcal{H}_j, \mathcal{H}_{j+1})$. The number of vector errors e_v in $\mathbf{H}$ against $\mathbf{H}^*$ are defined as:

$$e_v(\mathbf{H}) = \min_{\mathcal{M} \text{ is a matching sequence}} T_c(\mathcal{M}).$$

The number of vector errors e_v can be calculated by a dynamic programming algorithm in time $O(n(k-1)!^2)$, which is faster than the method used in (Berger *et al.*, 2014) with time complexity $O(kn^2)$, when k is small and n is big. Please see Figure S3 of the Supplementary Materials for the the pseudocode of the dynamic programming algorithm.

The vector error rate R_v is defined as e_v/n. For example, in Figure 3, there is a matching sequence $\mathcal{M} = \{\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3, \mathcal{H}_4\}$, where $\mathcal{H}_1 = \{H_1 \to H_2^*, H_2 \to H_1^*, H_3 \to H_3^*\}$ and $\mathcal{H}_2 = \mathcal{H}_3 = \mathcal{H}_4 = \{H_1 \to H_3^*, H_2 \to H_1^*, H_3 \to H_2^*\}$. It is easy to see that $T_c(\mathcal{M}) = 2$, the number of vector errors in $\mathbf{H}$ is 2 and the vector error rate $R_v = 2/4 = 0.5$.

To choose an appropriate weight w for H-PoPG, we tested its performance with different w from 0.8 to 1.0, and compared it with HapTree and HapCompass. We used the first contig of AWRI1499 as the haplotype template and ran ART with $c = 2$ to generate 100 triploid read data sets (each consisting of a set of single-end reads and a set of paired-end reads). After alignment to the contig and deleting alleles at homozygous SNP loci and reads that cover fewer than 2 heterozygous SNP loci, we obtained 100 SNP matrices, each of which has 449 columns

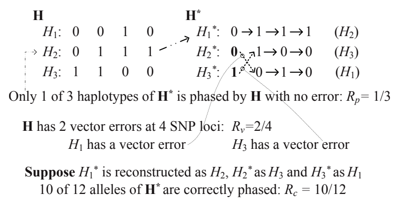
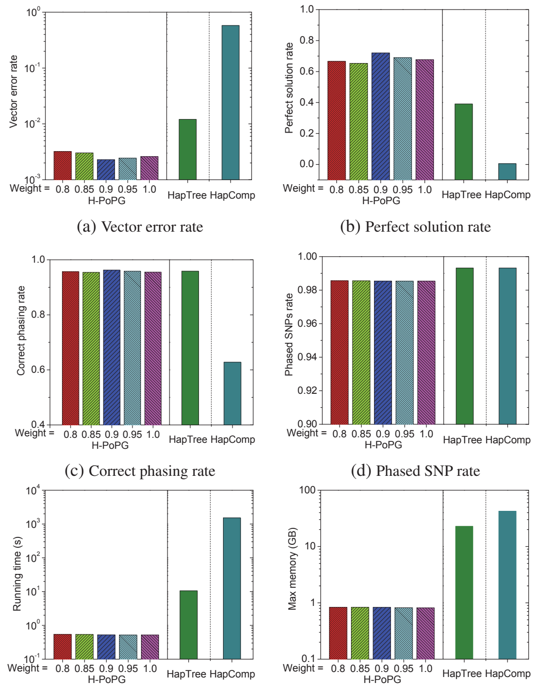

H
H*
H_1: 0 0 1 0
H_2: 0 1 1 1
H_3: 1 1 0 0
H_1^*: 0→1→1→1 (H_2)
H_2^*: 0→1→0→0 (H_3)
H_3^*: 1→0→1→0 (H_1)
Only 1 of 3 haplotypes of H* is phased by H with no error: R_p= 1/3
H has 2 vector errors at 4 SNP loci: R_v=2/4
H_1 has a vector error
H_3 has a vector error
Suppose H_1^* is reconstructed as H_2, H_2^* as H_3 and H_3^* as H_1
10 of 12 alleles of H* are correctly phased: R_c = 10/12

Fig. 3. An simple illustration of correct phasing rate, perfect solution rate and vector error rate. In this example, the correct phasing rate R_c of the reconstructed haplotypes H is 10/12, the perfect solution rate R_p = 1/3 and the vector error rate $R_v = 2/4$.



(a) Vector error rate (b) Perfect solution rate

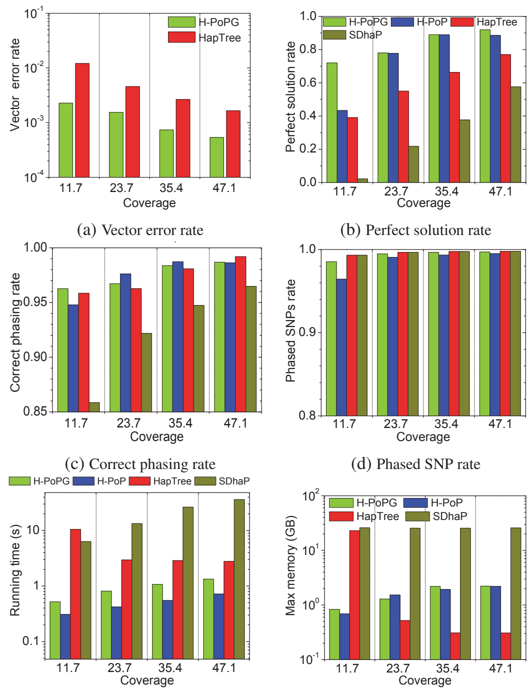
(c) Correct phasing rate (d) Phased SNP rate

(e) Running time (f) Max resident memory

Fig. 4. Comparison of the performance of H-PoPG, HapTree and HapCompass on simulated triploid ($k = 3$) 454 GS FLX Titanium sequencing data. Various weights from 0.8 to 1.0 were considered for the parameter w in H-PoPG.

(heterozygous SNPs). The average number of rows (SNP reads) of an SNP matrix is 394.08, the average number of non-'-' values of each row (called the SNP read average length) is 13.4, and the average number of non-'-' values of each column (regarded as the SNP read coverage) is 11.7.

The test results are showed in Figure 4. When the weight w was set as 0.9, H-PoPG reached the best performance among the three algorithms according to Figure 4 (a) in vector error rate, Figure 4 (b) in perfect solution rate and Figure 4 (c) in correct phasing rate. Figure 4 (d) shows that the rate of phased SNPs (*i.e.* the phased SNPs to total SNPs ratio) of HapTree and HapCompass is the best at 99.3%, while the phased SNP rates of H-PoPG with different weights are near 98.5%. Figures 4 (e) and (f) show that the running time and memory of H-PoPG varied little as the weight changes and H-PoPG used much less time and memory than HapTree and HapCompass. In the following experiments, w was set as 0.9 for H-PoPG and H-PoP as the default value. Since HapCompass ran very slow and its performance was clearly inferior to H-PoPG and HapTree, it was not included in the tests.



(a) Vector error rate (b) Perfect solution rate

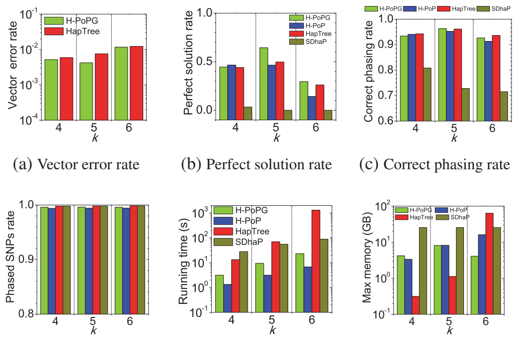
(c) Correct phasing rate (d) Phased SNP rate

(e) Running time (f) Max resident memory

Fig. 5. Comparison of the performance of H-PoPG, HapTree and SDhaP on simulated triploid ($k = 3$) 454 GS FLX Titanium sequencing data. The SNP read coverage increased from 11.7 to 23.7, 35.4 and 47.1, and the average SNP length was about 13.5.

We increased c from 2 to 4, 6 and 8, and generated SNP matrices with SNP read coverage changing from 11.7 to 23.7, 35.4 and 47.1 to test H-PoPG, H-PoP, HapTree and SDhaP. Figure 5 presents the test results. Without genotype information, the genotypes of the k haplotypes reconstructed by SDhaP and H-PoP were often different from that of the real k-haplotypes and hence vector error rate could not be used to measure the performance of SDhaP and H-PoP. When the coverage increased, the vector error rates, the perfect solution rates and the correct phasing rate of all algorithms improved. The vector error rates of H-PoPG are less than a half of those of HapTree, and the perfect solution rates of H-PoPG and H-PoP are clearly higher than those of HapTree and SDhaP. In the test with SNP read coverage 47.1, HapTree reached the highest correct phasing rate, while in the other tests H-PoPG performed best in term of correct phasing rate. HapTree and SDhaP phased the most SNPs, while the phased SNPs of H-PoPG were a little fewer than those of HapTree and SDhaP. In the test with SNP read coverage 11.7, the phased SNP rate of H-PoP was the lowest. However, even in the worst case, H-PoP phased more than 96% of the SNPs. In terms of efficiency, the average running times of H-PoPG and H-PoP were less than 2 seconds, which is obviously less than those of HapTree and SDhaP. H-PoPG and H-PoP used much less memory than HapTree and SDhaP when the coverage was 11.7, and HapTree used the least amount of memory in the other three cases. The memory requirement of SDhaP was about 26 GB, and it did not change much in different tests (including the tests below), while the memory required by H-PoPG and H-PoP was less than 3 GB in all four cases. It is interesting to observe that HapTree spent much more time and memory to handle the test data with coverage 11.7 than other data with deeper coverages.

We varied k from 4 to 6 to test the performance of the algorithms on reconstructing the haplotypes of a tetraploid, pentaploid or hexaploid



(a) Vector error rate (b) Perfect solution rate (c) Correct phasing rate

(d) Phased SNPrate (e) Running time (f) Max resident memory

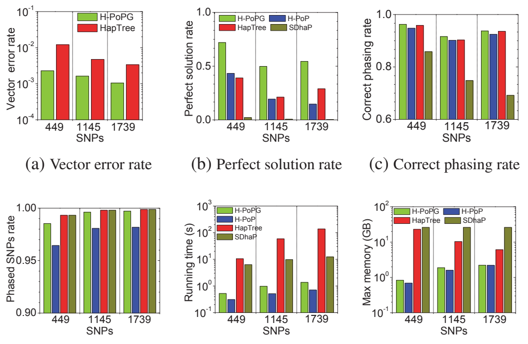
Fig. 6. Comparison of the performance of H-PoPG, H-PoP, HapTree, and SDhaP on simulated tetraploid ($k = 4$), pentaploid ($k = 5$) and hexaploid ($k = 6$) data. The number of SNPs was 449, and the average SNP read length was 13.5. The SNP read coverage of the simulated data with $k = 4(5, 6)$ was 47.1 (58.9, 70.8, respectively).

genome. Figure 6 illustrates the test results on simulated data generated by ART with $c = 6$. When $k = 5$, H-PoPG was superior to the other algorithms in terms of vector error rate, perfect solution rate and correct phasing rate. When $k = 4$ and 6, HapTree was the best in terms of vector error rate and correct phasing rate, but H-PoPG was not far behind. On the other hand, the running time and memory requirement of HapTree increased at a significantly faster rate than those of the other algorithms when k increased. When $k = 6$, HapTree ran for 1301 seconds and used 63 GB memory, while H-PoPG used only 23 seconds and 4 GB memory. As for the phased SNP rate, there is very little difference between the algorithms, and it was above 99.3% for all algorithms.

Figure 7 shows the test results on triploid data when we changed the haplotype template from the first contig of AWRI1499 to the concatenations of the first two contigs or the first four contigs. The number of heterozygous SNPs of the haplotype template increased from 449 to 1145 and 1739, respectively. Figure 7 illustrates that H-PoPG had the best performance in terms of vector error rate, perfect solution rate and correct phasing rate, while SDhaP was the worst in terms of perfect solution rate and correct phasing rate. The phased SNP rates of HapTree and SDhaP are a little higher than those of H-PoPG. The running time of HapTree increased significantly with the increased haplotype template length, while the running time of H-PoPG and H-PoP increased slowly. The running times and memories of H-PoPG and H-PoP are much less than those of HapTree and SDhaP.

The length of reads generated by ART is limited by the build-in sequencing platform quality profile. To test the performance of the algorithms on long reads that future sequencing technologies might produce, we concatenated multiple copies of the Illumina HiSeq 2500 platform quality profile contained in the ART package and generated a simulated future sequencer quality profile file to avoid the read length limit of ART. We concatenated all 14 contigs of the first scaffold of AWRI1499 into a template haplotype consisting of 22178 heterozygous SNPs, and using the new quality profile, we ran ART to generate reads with parameters r (the length of reads to be simulated), f (the mean length of inserts for paired-end reads), σ (the standard deviation of DNA fragment sizes), and c (the read coverage of each haplotype). We fixed c as 12, σ as 50, and varied (r, f) from (500, 2000) to (1000, 3000) and (2000, 6000), and the average length l of the generated SNP reads changed from 40 to 80 and 160.

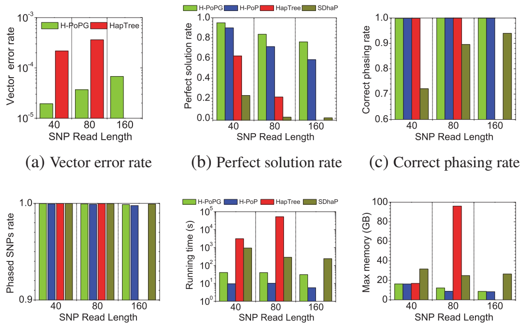
Figure 8 shows the test results. Since HapTree ran slowly when tested on the data with l = 80 and l = 160, in each case we divided 100 test datasets into 10 subsets and tested HapTree on 10 different nodes of the



(a) Vector error rate (b) Perfect solution rate (c) Correct phasing rate

(d) Phased SNP rate (e) Running time (f) Max resident memory

Fig. 7. Comparison of performance of H-PoPG, H-PoP, HapTree and SDhaP on simulated triploid ($k = 3$) 454 GS FLX Titanium sequencing data with the number of SNPs changed. The average SNP read length and coverage of the simulated data with 449 (1145, 1739) SNPs were 13.4 (14.1, 14.7) and 11.7 (11.9, 11.9), respectively.



(a) Vector error rate (b) Perfect solution rate (c) Correct phasing rate

(d) Phased SNP rate (e) Running time (f) Max resident memory

Fig. 8. Comparison of performance of H-PoPG, H-PoP, HapTree and SDhaP on simulated triploid ($k = 3$) sequencing data with different SNP read lengths. Each haplotype contained 22178 heterozygous SNPs. When the average SNP read length was 40 (80, 160), the average number of SNP reads and the SNP read coverage were 19719.2 (9827.7, 4877.3) and 35.6 (35.5, 35.2), respectively.

Linux cluster, each with a subset. As for the test with $l = 80$, since HapTree could only finish 4 subsets in 7 days, the results of HapTree in Figure 8 are the averages of its results on the 4 subsets. As for the test with l = 160, HapTree was unable to terminate on any subset in 7 days and hence its results are missing in Figure 8. Figure 8 shows again that H-PoPG is the best in terms of vector error rate and perfect phasing rate. The correct phasing rates of H-PoPG, H-PoP and HapTree were all more than 99.8%, while those of SDhaP were less than 94%. With regard to running time and memory, H-PoPG and H-PoP were clearly more efficient than the other two algorithms.

4 Conclusion

With the rapid development of sequencing technologies, reconstructing the multiple haplotypes of a polyploid from DNA sequencing reads is becoming more and more practical. In this paper, we modeled polyploid haplotyping as a combinatorial optimization problem to partition the input reads, called the Polyploid Balanced Optimal Partition (PBOP) problem. Since the problem is NP-hard, we developed a heuristic algorithm H-PoP for it. When the genotype information is available, we also proposed a genotype constrained version, called PBOPG, of the problem and designed a corresponding heuristic algorithm H-PoPG. Note that these methods are very different from our previous methods for dealing with diploids (Xie *et al.*, 2012) since they consider the distance between the consensus haplotypes of different groups while the previous methods for diploids calculates the distance between reads belonging to different groups. We compared our algorithms with three recent state-of-the-art polyploid haplotyping algorithms SDhaP, HapTree and HapCompass on both simulated and real data. Our extensive test results showed that H-PoPG was generally more accurate in reconstructing haplotypes than SDhaP, HapTree and HapCompass, and H-PoP was able to achieve comparable (though slightly worse) performance without the genotype information. Furthermore, H-PoPG and H-PoP ran much faster than SDhaP, HapTree and HapCompass, and could effectively handle long reads and deep read coverage.

Our experiments on real data also showed that H-PoP could be used to infer the number of chromosomes of an organism (*i.e.* its ploidy), since when the parameter k is set smaller than the ploidy of the organism, the MEC rate of H-PoP would be much larger than the sequencing error rate.

Funding

This work has been supported in part by the National Natural Science Foundation of China under grant NO. 61370172 and US National Science Foundation grant DBI-1262107.

References

Aguiar, D. and Istrail, S. (2013). Haplotype assembly in polyploid genomes and identical by descent shared tracts. *Bioinformatics*, **29**(13), i352–60.

Bafna, V., Istrail, S., Lancia, G., and Rizzi, R. (2005). Polynomial and APX-hard cases of the individual haplotyping problem. *Theoretical Computer Science*, **335**(1), 109–125.

Berger, E., Yorukoglu, D., Peng, J., and Berger, B. (2014). HapTree: a novel Bayesian framework for single individual polyplotyping using NGS data. *PLoS Comput Biol*, **10**(3), e1003502.

Bonizzoni, P., Dondi, R., Klau, G. W., Pirola, Y., Pisanti, N., and Zaccaria, S. (2015). On the fixed parameter tractability and approximability of the minimum error correction problem. volume 9133 of LNCS, pages 100–113. Springer International Publishing.

Browning, S. R. and Browning, B. L. (2011). Haplotype phasing: existing methods and new developments. *Nat Rev Genet*, **12**(10), 703–14.

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

Cilibrasi, R., van Iersel, L., Kelk, S., and Tromp, J. (2007). The complexity of the single individual SNP haplotyping problem. *Algorithmica*, **49**(1), 13–36.

Curtin, C. D., Borneman, A. R., Chambers, P. J., and Pretorius, I. S. (2012). De-novo assembly and analysis of the heterozygous triploid genome of the wine spoilage yeast Dekkera bruxellensis AWRI1499. *PLoS One*, **7**(3), e33840.

Das, S. and Vikalo, H. (2015). SDhaP: haplotype assembly for diploids and polyploids via semi-definite programming. *BMC Genomics*, **16**, 260.

Duitama, J., Huebsch, T., McEwen, G., Suk, E.-K., and Hoehe, M. R. (2010). ReFHap: a reliable and fast algorithm for single individual haplotyping. In *Proceedings of the First ACM international Conference on Bioinformatics and Computational Biology*, pages 160–169, Niagara Falls, New York. ACM.

Genovese, L. M., Geraci, F., and Pellegrini, M. (2008). SpeedHap: an accurate heuristic for the single individual SNP haplotyping problem with many gaps, high reading error rate and low coverage. *IEEE/ACM Trans Comput Biol Bioinform*, **5**(4), 492–502.

He, D., Choi, A., Pipatsrisawat, K., Darwiche, A., and Eskin, E. (2010). Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, **26**(12), i183–90.

Huang, W., Li, L., Myers, J. R., and Marth, G. T. (2012). ART: a next-generation sequencing read simulator. *Bioinformatics*, **28**(4), 593–4.

Lancia, G., Bafna, V., Istrail, S., Lippert, R., and Schwartz, R. (2001). SNPs problems, complexity and algorithms. In F. M. auf der Heide, editor, *Proc. Ann. European Symp. on Algorithms (ESA)*, volume 2161 of *Lecture Notes in Computer Science*, pages 182–193, Berlin/Heidelberg. Springer.

Leitch, A. R. and Leitch, I. J. (2008). Genomic plasticity and the diversity of polyploid plants. *Science*, **320**(5875), 481–3.

Li, H. (2011a). Improving SNP discovery by base alignment quality. *Bioinformatics*, **27**(8), 1157–8.

Li, H. (2011b). A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, **27**(21), 2987–93.

Li, H. and Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler transform. *Bioinformatics*, **25**(14), 1754–60.

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., Marth, G., Abecasis, G., and Durbin, R. (2009). The sequence alignment/map format and samtools. *Bioinformatics*, **25**(16), 2078–9.

Lippert, R., Schwartz, R., G.Lancia, and Istrail, S. (2002). Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Brief. Bioinform*, **3**(1), 1–9.

Margulies, M., Egholm, M., Altman, W. E., Attiya, S., *et al.* (2005). Genome sequencing in microfabricated high-density picolitre reactors. *Nature*, **437**(7057), 376–80.

Panconesi, A. and Sozio, M. (2004). Fast hare: a fast heuristic for single individual SNP haplotype reconstruction. In I. Jonassen and J. Kim, editors, *Proc. WABI*, volume 3240 of *Lecture Notes in Computer Science*, pages 266–277, Berlin/Heidelberg. Springer.

Patterson, M., Marschall, T., Pisanti, N., van Iersel, L., Stougie, L., Klau, G. W., and Schonhuth, A. (2015). WhatsHap: weighted haplotype assembly for future-generation sequencing reads. *J Comput Biol*, **22**(6), 498–509.

Pirola, Y., Zaccaria, S., Dondi, R., Klau, G. W., Pisanti, N., and Bonizzoni, P. (2015). HapCol: accurate and memory-efficient haplotype assembly from long reads. *Bioinformatics*, 10.1093/bioinformatics/btv495.

Renny-Byfield, S. and Wendel, J. F. (2014). Doubling down on genomes: polyploidy and crop plants. *Am J Bot*.

Wang, J., Xie, M., and Chen, J. (2010). A practical exact algorithm for the individual haplotyping problem MEC/GI. *Algorithmica*, **56**(3), 283–296.

Wang, R. S., Wu, L. Y., Li, Z. P., and Zhang, X. S. (2005). Haplotype reconstruction from SNP fragments by minimum error correction. *Bioinformatics*, **21**(10), 2456–2462.

Xie, M., Wang, J., and Chen, J. (2008). A model of higher accuracy for the individual haplotyping problem based on weighted SNP fragments and genotype with errors. *Bioinformatics*, **24**(13), i105–13.

Xie, M., Wang, J., Chen, J., Wu, J., and Liu, X. (2010a). Computational models and algorithms for the single individual haplotyping problem. *Current Bioinformatics*, **5**(1), 18–28.

Xie, M., Wang, J., and Chen, J. (2010b). A practical parameterised algorithm for the individual haplotyping problem MLF. *Mathematical Structures in Computer Science*, **20**(5), 851–863.

Xie, M., Wang, J., and Jiang, T. (2012). A fast and accurate algorithm for single individual haplotyping. *BMC Systems Biology*, **6**(Suppl 2), S8.