BIOINFORMATICS ORIGINAL PAPER

Vol. 29 no. 16 2013, pages 1938–1945
doi:10.1093/bioinformatics/btt349

Genome analysis

Advance Access publication June 18, 2013

Exact algorithms for haplotype assembly from whole-genome sequence data

Zhi-Zhong Chen[1,*], Fei Deng[2] and Lusheng Wang[2,*]

[1]Division of Information System Design, Tokyo Denki University, Saitama 350-0394, Japan and [2]Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Hong Kong

Associate Editor: Michael Brudno

ABSTRACT

Motivation: Haplotypes play a crucial role in genetic analysis and have many applications such as gene disease diagnoses, association studies, ancestry inference and so forth. The development of DNA sequencing technologies makes it possible to obtain haplotypes from a set of aligned reads originated from both copies of a chromosome of a single individual. This approach is often known as haplotype assembly. Exact algorithms that can give optimal solutions to the haplotype assembly problem are highly demanded. Unfortunately, previous algorithms for this problem either fail to output optimal solutions or take too long time even executed on a PC cluster.

Results: We develop an approach to finding optimal solutions for the haplotype assembly problem under the minimum-error-correction (MEC) model. Most of the previous approaches assume that the columns in the input matrix correspond to (putative) heterozygous sites. This all-heterozygous assumption is correct for most columns, but it may be incorrect for a small number of columns. In this article, we consider the MEC model with or without the all-heterozygous assumption. In our approach, we first use new methods to decompose the input read matrix into small independent blocks and then model the problem for each block as an integer linear programming problem, which is then solved by an integer linear programming solver. We have tested our program on a *single* PC [a Linux (x64) desktop PC with i7-3960X CPU], using the filtered HuRef and the NA 12878 datasets (after applying some variant calling methods). With the all-heterozygous assumption, our approach can optimally solve the whole HuRef data set within a total time of 31 h (26 h for the most difficult block of the 15th chromosome and only 5 h for the other blocks). To our knowledge, this is the first time that MEC optimal solutions are completely obtained for the filtered HuRef dataset. Moreover, in the general case (without the all-heterozygous assumption), for the HuRef dataset our approach can optimally solve all the chromosomes except the most difficult block in chromosome 15 within a total time of 12 days. For both of the HuRef and NA12878 datasets, the optimal costs in the general case are sometimes much smaller than those in the all-heterozygous case. This implies that some columns in the input matrix (after applying certain variant calling methods) still correspond to false-heterozygous sites.

Availability: Our program, the optimal solutions found for the HuRef dataset available at http://rnc.r.dendai.ac.jp/hapAssembly.html.

Contact: zzchen@mail.dendai.ac.jp or cswangl@cityu.edu.hk

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

*To whom correspondence should be addressed.

Received on February 13, 2013; revised on June 12, 2013; accepted on June 13, 2013

1 INTRODUCTION

A haplotype is the sequence of SNPs in each of the two copies of a given chromosome in a diploid organism. Haplotypes are crucial for genetic analysis and have many applications such as gene disease diagnoses, association studies, ancestry inference, drug design and so forth (Beckmann, 2010; Clark *et al.*, 1998; Hoehe *et al.*, 2000; Schwartz *et al.*, 2002).

Traditional approaches to obtaining haplotypes are based on genotype data from a set of individuals. The genotype data tell the status of both alleles at a position without distinguishing which one is on a specific copy of the chromosome. This approach is generally known as haplotype phasing. One can use various algorithms to infer the haplotypes (Eskin *et al.*, 2003; Halperin *et al.*, 2004; Stephens *et al.*, 2001; Wang and Xu, 2003). A drawback of this approach lies in its weakness in identifying rare and novel SNPs (He *et al.*, 2010). Besides, it is hard to verify whether the inferred haplotype is completely correct. With the development of high-throughput sequencing technologies, an alternative way to obtain the haplotypes for an individual is to combine sequence fragments, which is known as haplotype assembly (Chen *et al.*, 2008; Geraci, 2010; Li *et al.*, 2004; Panconesi and Sozio, 2004; Schwartz, 2010; Wang *et al.*, 2010; Xie *et al.*, 2012). Given a set of aligned reads sequenced from the two copies of a given chromosome of a single individual, the goal of haplotype assembly is to correctly determine two haplotypes, each of which corresponding to one of the two copies of the chromosome.

The haplotype assembly problem was first introduced by Lancia *et al.* (2001). Basically, when reads contain errors, the reads cannot be partitioned perfectly into two disjoint sets each of which consists of non-conflicting reads. To deal with errors when looking for the best reconstruction of haplotypes, one has to fix an objective function for evaluating candidate haplotypes. For this purpose, various functions such as Minimum Fragment Removal, Minimum SNP Removal, Longest Haplotype Reconstruction, Minimum Error Correction (MEC), Minimum Implicit SNP Removal and Minimum Implicit Fragment Removal have been subsequently proposed (Lancia *et al.*, 2001; Lippert *et al.*, 2002). Recently, Aguiar and Istrail (2012) proposed the Minimum Weighted Edge Removal function, whereas Duitama *et al.* (2010, 2012) proposed the Maximum Fragments Cut function. Of special interest among the proposed functions is

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

© The Author 2013. Published by Oxford University Press. All rights reserved. For Permissions, please email: journals.permissions@oup.com

MEC, which aims at minimizing the total number of conflicts (errors) between the reads and the constructed haplotypes (h_1, h_2). The problem of minimizing MEC is NP hard (Cilibrasi *et al*., 2005; Lippert *et al*., 2002). For this problem, Wang *et al*. (2005) presented an exact algorithm based on the branch-and-bound method and a genetic algorithm. A weighted version of this problem is considered by Zhao *et al*. (2005). In the remainder of this article, we only consider the problem of minimizing MEC.

Levy *et al*. (2007) presented the first diploid genome sequence of an individual human, J. Craig Venter, using Sanger sequencing technology. They also designed a greedy heuristic method that concatenates the reads with minimum conflicts. Their method is fast but not accurate when errors appear in reads. Bansal and Bafna (2008) developed a software package (named HapCUT), and their algorithm tries to minimize the MEC score of the reconstructed haplotypes by iteratively computing max-cuts in graphs derived from the sequenced fragments. Bansal *et al*. (2008) designed a Markov chain Monte Carlo algorithm (named HASH). Both HASH and HapCut work well in practice, but there is no guarantee of finding optimal haplotypes.

Recently, He *et al*. (2010) proposed a dynamic programming algorithm for the problem that runs in time $O(2^k mn)$, where k is the length of the longest read, m is the number of reads and n is the total number of SNPs in the haplotypes. Their experiments show that their algorithm works well when $k \leq 15$. On the other hand, when k is large, they model the problem as a MaxSAT problem, which is then solved by a MaxSAT solver. To compare their MaxSAT approach with the previous methods, they use the filtered HuRef dataset from Levy *et al*. (2007) over 22 chromosomes. Via experiments, they show that their program can construct better haplotypes than the previous programs by Bansal and Bafna (2008) and Levy *et al*. (2007). It is worth pointing out that to solve the problem for the 22 chromosomes, their program takes a total time of ~15 h on a *PC cluster*. Moreover, their program does not solve the problem exactly because it excludes certain reads (3725 reads in total) from consideration. Furthermore, their program fails to find optimal haplotypes for a total of eight blocks of the 22 chromosomes.

In this article, we develop a new approach to optimally solving the problem. In our approach, we first use new methods to decompose the input read matrix into small independent blocks and then model the problem for each block as an integer linear programming (ILP) problem, which is then solved by an ILP solver [such as CPLEX (IBM ILOG CPLEX Optimizer) and GLPK(GNU Linear Programming Kit)]. We have tested our program on a *single* PC [namely, a Linux (x64) desktop PC with i7-3960X CPU], using the filtered HuRef dataset. Our experimental results show that our program can optimally solve all the chromosomes within a total time of 31 h (26 h for the most difficult block of the 15th chromosome and only 5 h for the other blocks). To our knowledge, this is the first time that optimal haplotypes under the MEC model are completely obtained for the filtered HuRef dataset. Moreover, to find almost optimal haplotypes within much shorter time for the difficult blocks, we propose several powerful heuristic methods.

Wu *et al*. (2009) have generalized the problem by removing the *all-heterozygous assumption* to handle the existence of a small number of homozygous sizes in the solution. The generalized problem is much harder because it allows many more candidate haplotypes. Nevertheless, we develop a program that can optimally solve the generalized problem for all the 22 chromosomes of the filtered HuRef dataset except the most difficult block of the 15th chromosome within a total time of 12 days. As far as we know, this is the first strike on computing optimal solutions for the HuRef dataset without the all-heterozygous assumption. Moreover, to find almost optimal haplotypes within much shorter time for the difficult blocks, we propose several powerful heuristic methods. Via experiments with the simulated dataset of Geraci (2010), we show that an optimal solution for the general case of the problem achieves a better reconstruction rate than an optimal solution for the all-heterozygous case of the problem.

2 METHODS

2.1 The haplotype assembly problem

For convenience, we define a *ternary string* to be a string whose characters each belong to $\{0, 1, -\}$. The *extended Hamming distance* between two ternary strings s and t, denoted by $d(s, t)$, is the total number of positions p at which both characters of s and t belong to $\{0, 1\}$, but they are different. Two binary strings h and h' of the same length are *complementary* if the bit of h at every position is different from the bit of h' at the same position.

In the *haplotype assembly problem*, we are given a matrix of X whose entries each belong to $\{0, 1, -\}$ (i.e. each row of X is a ternary string). Supplementary Table S1 shows an example X. Such an X is constructed from a given reference genome sequence and the set of reads containing sequence from both chromosomes, by aligning all the reads to the reference genome and then applying some SNP/variant calling methods. See Section 2 of the Supplementary Material for details. Each row of X corresponds to a read, whereas each column corresponds to an SNP site. The first (last, respectively) entry of a read that is not a '−' is called the *start* (*end*, respectively) *position* of the read. There can exist '−'s between the start and the end positions of a read. Such '−'s are called *gaps* of the read because they either correspond to missing data or serve as gaps to connect disjoint parts of the read. If a read has no gap, it is called a *gapless* read; otherwise, it is called a *gapped* read. Moreover, if the gaps of a gapped read appear consecutively, then it is called a *paired-end* read; otherwise, it is called a *multi-gapped* read. The *length* of a read is $j_e - j_s + 1$, where j_e and j_s are the end and the start positions of the read, respectively.

Given X, we want to compute the unknown haplotypes, which are an unordered pair $H = (h, h')$ of binary strings each of length n, where n is the number of columns in X. Such a pair is called a *solution* of X. As the haplotypes are unknown and there are many solutions for them, we need a criterion for evaluating solutions. The MEC score is such a criterion and is defined as follows: Given X, the MEC score of a solution $H = (h, h')$ of X is $\sum_{i=1}^{m} \min\{d(r_i, h), d(r_i, h')\}$, where m is the number of rows in X and r_i is the ith row of X. Given X, the haplotype assembly problem asks for a solution of X whose MEC score is minimized. Such a solution is called an *optimal solution* of X.

For convenience, we say that a row of X is *useless* if every entry in the row is a '−' and is *useful* otherwise. Clearly, the removal of useless rows from X does not change the problem. Moreover, we say that a column of X is *monotone* if 0 or 1 does not appear in the column. Obviously, if the jth column of X contains no 0's (1's, respectively), then X has an optimal solution (h, h') such that the jth bits of h and h' are 1's (0's, respectively). Recall that a diploid organism has two alleles at each position. An SNP site is *homozygous* if the two alleles at this site are identical; otherwise, it is *heterozygous*. When the reads contain no errors, a homozygous SNP site corresponds to a monotone column in X, whereas a heterozygous SNP site corresponds to a non-monotone column in X. However, when reads

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

contain errors at a homozygous SNP site, the corresponding column can also be non-monotone. Therefore, we may hereafter make the following assumptions:

A1. No row of X is useless.
A2. No column of X is monotone.
A3. No column of X contains more 1's than 0's.

We make Assumption A3 only for a technical reason. If the jth column of X contains more 1's than 0's, then we can modify the column by flipping 1's and 0's so that each solution (h, h') of the modified X can be transformed back to a solution of the original X with the same MEC score by flipping the jth bits of h and h'. Therefore, it does not matter to instead assume that no column of X contains more 0's than 1's.

Assumption A2 ensures that all the columns of X contain both 0's and 1's. However, some originally homozygous sites may also be included in X owing to errors in the reads.

Given a solution (h, h') of X, we can obtain a bipartition (H, H') of the rows in X as follows. Each row r_i of X with $d(r_i, h) \leq d(r_i, h')$ belongs to H, whereas each row r_i of X with $d(r_i, h) > d(r_i, h')$ belongs to H'. Consider an arbitrary integer $j \in \{1, 2, \ldots, n\}$. Let $h_{j,0}$ ($h_{j,1}$, respectively) be the number of rows r_i in H such that the entry in the jth column of r_i is a 0 (1, respectively). Similarly, let $h'_{j,0}$ ($h'_{j,1}$, respectively) be the number of rows r_i in H' such that the entry in the jth column of r_i is a 0 (1, respectively). We define the *contribution* of the jth column of X to the MEC score of (h, h') as follows: Let h_j and h'_j be the jth bit of h and h', respectively. If $h_j h'_j$ is 00 (11, 01 or 10, respectively), then the contribution is $h_{j,1} + h'_{j,1}$ ($h_{j,0} + h'_{j,0}$, $h_{j,1} + h'_{j,0}$ or $h_{j,0} + h'_{j,1}$, respectively). The MEC score of (h, h') is the total contribution of the columns of X.

2.2 The all-heterozygous case

Recall that each column of X is not monotone based on Assumption A2. When there is no error, every non-monotone column in X should correspond to a heterozygous site in the solution. When errors occur, it is still true that most of the columns in X correspond to heterozygous sites in the solution. This motivates researchers to assume that all sites corresponding to the columns in X are heterozygous, i.e. we should search for solutions (h, h') of X such that h and h' are complementary (He *et al.*, 2010). In this subsection, we show how to solve this case by reductions and ILP.

For each pair of integers (j_1, j_2) with $1 \leq j_1 \leq j_2 \leq n$, we use $X[j_1, j_2]$ to denote the submatrix of X consisting of the j_1th, $(j_1 + 1)$st, ..., j_2th columns of X. For brevity, we simply use $X[j]$ to denote $X[j, j]$.

Block reduction: Like most of the existing methods for the problem, we split the whole chromosome into a set of column-disjoint blocks such that no read starts and ends at different blocks. This reduction can be done in time linear in the total length of reads in X.

Many rows of the blocks are useless and can be removed immediately. A *singular* block is a useful block consisting of a single column. Singular blocks can be ignored because the problem for them is trivially solved.

Block decomposition: Suppose that B is a block of X with ℓ columns. Consider an integer j with $1 < j < \ell$. If there is no read r in B such that j is greater than the start position of r but less than the end position of r, then we say that $B[j]$ is a *splittable* column. Suppose that the splittable columns of B are $B[j_1]$, $B[j_2]$, ..., $B[j_k]$. Then, the submatrices $B[1, j_1]$, $B[j_1, j_2]$, $B[j_2, j_3]$, ..., $B[j_{k-1}, j_k]$, $B[j_k, \ell]$ are called *unsplittable blocks* of X. Many rows of these blocks are useless and can be removed immediately. An important observation is that we can solve the problems for these blocks independently (say, in parallel) and then concatenate their optimal solutions $(h_1, h'_1), (h_2, h'_2), \ldots, (h_{k+1}, h'_{k+1})$ into an optimal solution (h, h') of B as follows. Consider an integer i with $1 \leq i \leq k$. As h_i and h'_i are complementary, one of them ends with a 0 and the other ends with a 1. Similarly, one of h_{i+1} and h'_{i+1} starts with a 0 and the other starts with a 1. Therefore, the last character of h_i is the same as the first character of h_{i+1} or h'_{i+1}. We assume that the last character of h_i is the same as the first character of h_{i+1}; the other case is similar. Then, we can concatenate (h_i, h'_i) and (h_{i+1}, h'_{i+1}) by (1) deleting the last character of h_i and then appending h_{i+1} to h_i and (2) deleting the last character of h'_i and then appending h'_{i+1} to h'_i. This shows that we can reduce the original problem for B to the smaller problems for $B[1, j_1]$, $B[j_1, j_2]$, $B[j_2, j_3]$, ..., $B[j_{k-1}, j_k]$, $B[j_k, \ell]$. We call this reduction *block decomposition*. It can be done in time linear in the total length of reads in B.

Singleton removal: Suppose that C is an unsplittable block of X. A row in C is *singular* if the start and the end positions of the read corresponding to the row are the same. Obviously, the removal of a singular row from C does not change the MEC score of an optimal solution of C. Therefore, we can reduce the problem for C to a smaller problem by removing all singular rows from C. This type of reduction is called *singleton removal* and can be done in time linear in the total length of reads in C.

Duplicate removal: Suppose that C is an unsplittable block of X. *Duplicate removal* is the reduction that repeats modifying C as follows, until no two rows or columns of C are identical:

(1) Merge identical rows into a single row and memorize the original multiplicity of the row. (Of course, if a row is originally identical to no other row, then its multiplicity is 1.)
(2) Merge identical columns into a single column and memorize the original multiplicity of the column. (Of course, if a column is originally identical to no other column, then its multiplicity is 1.)

Duplicate removal can be done in $O(L \log k)$ time, where k is the number of reads in C and L is the total length of reads in C.

Solving reduced-blocks via ILP: A *reduced block* of X is an unsplittable block of X without singular rows, identical rows or identical columns. Suppose that D is a reduced block of X. To solve the problem for D, we formulate the problem as an ILP problem and then solve it using CPLEX of IBM, which is a freely available ILP solver for academic research.

Let p (q, respectively) be the number of rows (columns, respectively) of D. For each integer j with $1 \leq j \leq q$, let c_j be the multiplicity of $D[j]$. Similarly, for each integer i with $1 \leq i \leq p$, let w_i be the multiplicity of the ith row of D, and $J_{i,0}$ ($J_{i,1}$, respectively) be the set of integers $j \in \{1, 2, \ldots, q\}$ such that the ith entry in $D[j]$ is a 0 (1, respectively).

As we want to compute an optimal pair (h, h') of complementary haplotypes for D, we introduce a binary variable x_j for $D[j]$ whose value is supposed to be 1 if and only if the jth bit of h is a 1 (and hence the jth bit of h' is a 0). Moreover, we introduce a binary variable y_i for the ith row of D whose value is supposed to be 1 if and only if the read corresponding to r_i is aligned to h, where r_i is the ith row of D. Then, the problem of finding an optimal pair (h, h') of complementary haplotypes for D becomes the following integer programming problem:

$$\text{Minimize} \sum_{i=1}^{p} w_i \sum_{j \in J_{i,0}} c_j \big(x_j y_i + (1 - x_j)(1 - y_i)\big) + \sum_{i=1}^{p} w_i \sum_{j \in J_{i,1}} c_j \big((1 - x_j) y_i + x_j (1 - y_i)\big)$$

$$\text{Subject to } \forall_{1 \leq i \leq p}\ y_i \in \{0, 1\} \quad \forall_{1 \leq j \leq q}\ x_j \in \{0, 1\}$$

The aforementioned integer programming problem is not linear because it contains quadratic terms such as $x_j y_i$. Therefore, for each pair (i, j) with $1 \leq i \leq p$ and $1 \leq j \leq q$, we introduce a binary variable $t_{i,j}$ (for replacing $x_j y_i$) and add the following three constraints to ensure that $t_{i,j} = y_i x_j$:

$$t_{i,j} \leq y_i \quad t_{i,j} \leq x_j \quad t_{i,j} \geq y_i + x_j - 1$$

The resulting ILP formulation is given in the Supplementary Material. It is easy to construct the ILP from D in $O(Lp)$ time, where L is the total length of reads in D.

A powerful heuristic for hard blocks: Let C be an unsplittable block of X. Our experiments will show that performing singleton and duplicate

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

removal on C usually yields a reduced block D of X such that D is small enough that the aforementioned ILP problem for D can be solved by CPLEX within several seconds on a single PC. However, although rare, it is possible that the aforementioned ILP problem for D cannot be solved by CPLEX within several hours. In this case, we can solve the problem for C heuristically as follows.

(1) First, we choose an integer j with $1 \leq j < q$, where q is the number of columns in C. We refer to j as the *cut position*. The choice of the cut position is important, and we will detail how to choose it later.

(2) Then, we perform singleton and duplicate removal on $C[1, j]$ and $C[(j+1), q]$ and further use CPLEX to solve the smaller problems for the two resulting reduced blocks independently (say, in parallel) to obtain optimal solutions $(h_{j,1}, h'_{j,1})$ and $(h_{j,2}, h'_{j,2})$ for $C[1, j]$ and $C[(j+1), q]$. Let $s_{j,1}$ and $s_{j,2}$ be the MEC scores of the two solutions. Obviously, $s_{j,1} + s_{j,2}$ is a lower bound on the MEC score of an optimal solution of C.

(3) We concatenate the solutions obtained in Step 2 to obtain two solutions $(h_{j,1}h_{j,2}, h'_{j,1}h'_{j,2})$ and $(h_{j,1}h'_{j,2}, h'_{j,1}h_{j,2})$ of C. Unfortunately, it is often the case that neither solution has a MEC score close to the lower bound $s_{j,1} + s_{j,2}$. Therefore, we refer to them as the *raw solutions* for C associated with j and will design a method for refining them. Finally, among the two refined solutions, we output the one with the smaller MEC score.

For choosing the cut position, we propose the following two methods:

C1. For each $j \in \{1, \ldots, q-1\}$, we use an approximation algorithm to estimate the MEC score $s'_{j,1}$ ($s'_{j,2}$, respectively) of an optimal solution of $C[1, j]$ ($C[(j+1), q]$, respectively). We measure the quality of j by $|s'_{j,1} - s'_{j,2}|$. By trying all $j \in \{1, \ldots, q-1\}$, we can find the j with the best quality. The intuition behind this choice of the cut position is that if $s'_{j,1}$ and $s'_{j,2}$ are close, then it takes roughly the same amount of time to solve the problems for $C[1, j]$ and $C[(j+1), q]$. Thus, we refer to this choice as the *balanced choice* of the cut position. Basically, this choice aims at shortening the running time. Deng *et al.* (2013) have recently designed an approximation algorithm for estimating the MEC score.

C2. Let p be the number of rows in C. For each $j \in \{1, \ldots, q-1\}$, we find the set R_j of integers $i \in \{1, \ldots, p\}$ such that $f_i \leq j < t_i$, where f_i and t_i are the start and the end positions of the read corresponding to the ith row of C, respectively. For each $i \in R_j$, we also calculate $e_{i,j} = \min\{j - f_i + 1, t_i - j\}$. We further calculate $e_j = \sum_{i \in R_j} e_{i,j}$. For example, if C is the unsplittable block in the bottom of the matrix in Table 2 in the Supplementary Material, then $R_2 = \{3, 4\}$, $e_{3,2} = 2$, $e_{4,2} = 1$ and $e_2 = 3$. It is not hard to see that the MEC score of each raw solution of C associated with j does not exceed $e_j + s_{j,1} + s_{j,2}$, where $s_{j,1}$ and $s_{j,2}$ are the MEC scores of optimal solutions of $C[1, j]$ and $C[(j+1), q]$, respectively. Thus, $e_j + s_{j,1} + s_{j,2}$ is an upper bound on the MEC score of an optimal solution of C. Also recall that $s_{j,1} + s_{j,2}$ is a lower bound on the MEC score of an optimal solution of C. Hence, the smaller e_j is, the closer the two bounds are. Therefore, by trying all $j \in \{1, \ldots, q-1\}$, we can find the integer j that minimizes e_j. Unfortunately, such a j is usually either close to 1 or close to q and hence either the problem for $C[1, j]$ or the problem for $C[(j+1), q]$ remains hard to solve. Therefore, instead of trying all $j \in \{1, \ldots, q-1\}$, we choose a real number ε with $0 < \epsilon < 0.5$ (say, 0.1) and only try all $j \in \{\epsilon q, \ldots, (1-\epsilon)q\}$ to find the integer j with the smallest e_j. We refer to this choice as the *unbalanced choice* of the cut position.

For the purpose of refining a raw solution (h, h') of C associated with the cut position j, we design a subroutine (named *Majority*), which repeats the following steps until (h, h') does not change any more.

(a) For each read $i \in \{1, \ldots, p\}$, we compute $d(r_i, h)$ and $d(r_i, h')$, where r_i is the ith row of C. If $d(r_i, h) \leq d(r_i, h')$, we align r_i to h; otherwise, we align r_i to h'. In this way, each row is aligned to one of h and h'. Let S (S', respectively) be the set of rows aligned to h (h', respectively).

(b) For each $j \in \{1, \ldots, q\}$, we compute the total number $\eta_{j,0}$ ($\eta_{j,1}$, respectively) of rows $r_i \in S$ such that the jth entry in r_i is a 0 (1, respectively), and we also compute the total number $\eta'_{j,0}$ ($\eta'_{j,1}$, respectively) of rows $r_i \in S'$ such that the jth entry in r_i is a 0 (1, respectively). Let $b_j \in \{0, 1\}$ be the jth bit of h. Note that $1 - b_j$ is the jth bit of h'. Also note that $C[j]$ contributes $\eta_{j,1-b_j} + \eta'_{j,b_j}$ to the MEC score of (h, h'). Similarly, if we flip the jth bits of h and h', then $C[j]$ contributes $\eta_{j,b_j} + \eta'_{j,1-b_j}$ to the MEC score of (h, h'). Thus, we check whether $\eta_{j,1-b_j} + \eta'_{j,b_j} > \eta_{j,b_j} + \eta'_{j,1-b_j}$. If this inequality holds, then we flip the jth bits of h and h'.

Majority usually refines a raw solution (h, h') of C associated with the cut position j so that its MEC score becomes significantly smaller. To enhance *Majority*, we propose the following random-sampling approach. First, we choose a large enough number τ, say $\tau = 100 * s$, where s is the current MEC score of (h, h'). Then, we repeat the following three steps τ times:

(1) Calculate the current MEC score s of (h, h') and select (not necessarily distinct) integers $j_1, j_2, \ldots, j_\kappa$ each uniformly at random from $\{1, 2, \ldots, q\}$, where $\kappa = \lfloor 0.1s \rfloor$.

(2) Let $j'_1, j'_2, \ldots, j'_\ell$ be the distinct integers among $j_1, j_2, \ldots, j_\kappa$. Obtain a new solution $(\hat{h}, \hat{h}')$ of C by flipping the j'_i th bits of h and h' for all j'_i with $1 \leq i \leq \ell$.

(3) Call *Majority* to refine the solution $(\hat{h}, \hat{h}')$. If the MEC score of the refined $(\hat{h}, \hat{h}')$ is smaller than s, then update (h, h') to $(\hat{h}, \hat{h}')$.

2.3 The general case

Ideally, the input matrix X should only contain heterozygous sites for the specific *individual*. However, accurately identifying the set of heterozygous sites of an individual is extremely difficult. Most of the existing methods choose non-monotone columns to form X. This is based on the observation that when each read contains no errors, a non-monotone column corresponds to a heterozygous site and a monotone column corresponds to a homozygous site. However, in practice, reads may contain a (small) number of errors. Thus, some homozygous sites may also lead to non-monotone columns, and those non-monotone columns corresponding to homozygous sites are also included in the input matrix X. To deal with this problem, we have to consider the general case where each site in the solution (h, h') can be either homozygous or heterozygous. See the Supplementary Material for details.

In this case, neither block decomposition nor singleton removal is applicable anymore because some sites corresponding to the columns of X may be homozygous, and hence an optimal solution (h, h') of X may not be complementary. For each $j \in \{1, 2, \ldots, n\}$, we say that the jth column of X is *intrinsically heterozygous* if for every optimal solution (h, h') of X, we can modify the jth bits of h and h' without losing the optimality so that the jth bit of h is different from that of h'. An intrinsically heterozygous column is useful because as in the all-heterozygous case, it can be used to perform singleton removal and is also possibly a splittable column for block decomposition. We next consider how to find heterozygous columns.

Finding intrinsically heterozygous columns: For each $j \in \{1, 2, \ldots, n\}$, we compute the total number $\eta_{j,0}$ ($\eta_{j,1}$, respectively) of rows r_i in X such that the entry in the jth column of r_i is a 0 (1, respectively), and further compute the number $\bar{s}_{j,0}$ ($\bar{s}_{j,1}$, respectively) of non-singular rows r_i in X such that the jth entry in r_i is a 0 (1, respectively).

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

LEMMA 1. Suppose that $j \in \{1, 2, \ldots, n\}$ satisfies $\min\{\eta_{j,0}, \eta_{j,1}\} \geq \left\lfloor \frac{\tilde{s}_{j,0}+\tilde{s}_{j,1}}{2} \right\rfloor$. Then, $X[j]$ is intrinsically heterozygous.

PROOF. See Section 5 of the Supplementary Material.

For example, LEMMA 1 ensures that the first six columns of the matrix in Supplementary Table S2 are intrinsically heterozygous.

Block reduction: This is same as that in the all-heterozygous case.

Block decomposition: This is almost the same as that in the all-heterozygous case. The only difference is in the definition of a *splittable* column of a block B. Specifically, the new definition of a splittable column $B[j]$ of B requires that the jth column of B be intrinsically heterozygous.

Singleton removal: This is almost the same as that in the all-heterozygous case. The only difference is that instead of removing all singular rows, we only remove those singular rows r_i such that the read corresponding to r_i starts and ends at an intrinsically heterozygous column.

Duplicate removal: This is the same as that in the all-heterozygous case.

Solving reduced-blocks via ILP: Similar to the all-heterozygous case, we can obtain an ILP formulation for the general case. The only difference is that for each column $D[j]$ of D that is not known to be intrinsically heterozygous, we need to introduce *two* binary variables x_j and z_j (instead of one for the all-heterozygous case) such that the value of x_j (z_j, respectively) is supposed to be 1 if and only if the jth bit of h (h', respectively) is a 1. For lack of space, we detail the ILP formulation in the Supplementary Material.

It is worth pointing out that Wu *et al*. (2009) has also given an ILP formulation for the problem. However, their ILP formulation contains non-binary variables. Moreover, finding intrinsically heterozygous columns and merging identical columns enable us to use fewer variables than theirs. Consequently, ours can be solved within shorter time.

A powerful heuristic for hard blocks: This is almost the same as that in the all-heterozygous case. There are only two differences. The first is in Step (b) of the Majority subroutine, which should be modified as follows.

(b) For each $j \in \{1, \ldots, q\}$, we compute the total number $\eta_{j,0}$ ($\eta_{j,1}$, respectively) of rows $r_i \in S$ such that the entry in the jth column of r_i is a 0 (1, respectively), and we also compute the total number $\eta'_{j,0}$ ($\eta'_{j,1}$, respectively) of rows $r_i \in S'$ such that the entry in the jth column of r_i is a 0 (1, respectively). Let $b_j \in \{0,1\}$ (b'_j, respectively) be the jth bit of h (h', respectively). If $\eta_{j,1-b_j} > \eta_{j,b_j}$, then we flip the jth bit of h. Furthermore, if $\eta'_{j,1-b'_j} > \eta'_{j,b'_j}$, then we flip the jth bit of h'.

The other difference is in the random-sampling approach, whose last two steps should be modified as follows.

(1) Let $j'_1, j'_2, \ldots, j'_\ell$ be the distinct integers among $j_1, j_2, \ldots, j_\kappa$. Obtain a new solution $(\hat{h}, \hat{h}')$ of D by performing the following step for all j'_i with $1 \leq i \leq \ell$:

- If the j'_ith column of D is intrinsically heterozygous, then flip the j'_i th bits of h and h'; otherwise, choose one of h and h' uniformly at random and flip its j'_i th bit.

(2) Call the modified *Majority* subroutine to refine $(\hat{h}, \hat{h}')$. If the MEC score of the refined $(\hat{h}, \hat{h}')$ is smaller than s, then update (h, h') to $(\hat{h}, \hat{h}')$.

Another powerful heuristic for hard blocks: The ILP formulation of the all-heterozygous case has fewer variables and constraints than that of the general case and hence can usually be solved within much shorter time. Therefore, a natural heuristic for hard reduced-blocks D in the general case is as follows. First, we solve the problem for D by assuming that all columns of D are heterozygous. This yields a raw solution (h, h') of D. We then use the modified *Majority* subroutine and the random-sampling approach to refine (h, h'). Our experiments will show that this heuristic often finds a solution of D whose MEC score is extremely close to optimal. A drawback of this heuristic is that it does not give any lower bound on the MEC score of an optimal solution of D.

3 RESULTS AND DISCUSSION

To evaluate our methods empirically, we run our program on a Linux (x64) desktop PC with i7-3960X CPU and 31.4GiB RAM. In our experiments, we use three datasets.

3.1 The filtered HuRef dataset

The filtered HuRef dataset over 22 chromosomes is generated by Levy *et al*. (2007). Some simple variant calling method has been applied to form the matrices in the dataset (Levy *et al*., 2007). The dataset has been used to compare previous methods with each other (Bansal and Bafna, 2008; Bansal *et al*., 2008; He *et al*., 2010; Levy *et al*., 2007). This dataset is known to be hard to solve. Indeed, He *et al*. (2010)'s program takes 15 h on a PC cluster to only roughly solve the all-heterozygous case of the problem for the dataset. In particular, their program excludes multi-gapped reads (3725 in total) and fails to solve a total of eight hard blocks.

As each of the 22 chromosomes has a large number of SNP sites and a huge number of reads, it is effective to cut each of them into as many smaller independent blocks as possible. Block reduction has been used for this purpose in previous studies (He *et al*., 2010). In contrast, to our knowledge, block decomposition has not been used for this purpose before. For the all-heterozygous (general, respectively) case, Supplementary Table S6 (Supplementary Table S7, respectively) summarizes the numbers of non-singular blocks of the 22 chromosomes obtained by block reduction only and by both block reduction and block decomposition, respectively. As can be seen from the tables, block decomposition enables us to cut the nonsingular blocks (obtained by block reduction only) of a chromosome into many smaller blocks.

When we use CPLEX to solve each reduced-block in the all-heterozygous (general, respectively) case of the problem, we set a time limit of 10 (20, respectively) min. As the result, CPLEX fails to solve only three (seven, respectively) reduced-blocks optimally within the time limit. The three (seven, respectively) blocks are shown in the first (second, respectively) part of Supplementary Table S8. As they are hard, we use the (first) heuristic detailed in Section 2.2 (Section 2.3, respectively) to find heuristic solutions for them. The results are summarized in the same table from which we can see that our heuristics can find solutions extremely close to optimal for hard reduced-blocks. Excluding these hard blocks, our program can solve the all-heterozygous (general, respectively) case of the problem for the 22 chromosomes within a total of 3 h (5, respectively) on the PC. The running times and the optimal MEC scores for the 22 chromosomes for the all-heterozygous case and the general case are summarized in Table 1, where one can see that the scores for the general case are significantly smaller than those for the all-heterozygous case. This indicates that the number of homozygous sites included in the input matrix cannot be ignored.

In Table 1, we also compare our program with HapCUT (Bansal and Bafna, 2008). It is worth noting that HapCUT uses randomness and hence different runs of HapCUT on the

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

same input may generate different outputs. Therefore, for each chromosome in the HuRef dataset, we ran HapCUT 10 times, where each run was given the default maximum number (namely, 100) of iterations. Consequently, for each chromosome in the dataset, the running time of HapCUT in Table 1 is the total time of the 10 runs, whereas the score of HapCUT in the table is the best score among the 10 runs. Clearly, our program spends less time than HapCUT to give *optimal* solutions.

We also use the second heuristic in Section 2.3 to solve the general case of the problem for the seven hard reduced-blocks. The results are summarized in Table 2 from which we can see that the second heuristic can find solutions close to optimal for the hard reduced-blocks within shorter time than the first heuristic, but cannot find a lower bound on the optimal MEC score.

3.2 Simulated datasets

We use part of the simulated datasets of Geraci (2010). To generate a read matrix, three parameters ℓ, c and e are used, where ℓ is the number of SNPs, c is the coverage and e is the error rate. Intuitively speaking, the larger c (e, respectively) is, the more reads we have in the matrix (the larger optimal MEC score we have for the matrix, respectively). The values for ℓ, c and e are chosen from $\{100, 350, 700\}$, $\{3, 5, 8, 10\}$ and $\{0\%, 10\%, 20\%, 30\%\}$, respectively. For each combination of the three parameters, 100 read matrices are generated. A merit of each generated read matrix X is that we know its true solution (h_1, h_2). To evaluate the quality of a solution $(\hat{h}_1, \hat{h}_2)$ of X (returned by a program), the *reconstruction rate* of the solution is defined to be $1 - \frac{\min\{d(h_1,\hat{h}_1)+d(h_2,\hat{h}_2), d(h_1,\hat{h}_2)+d(h_2,\hat{h}_1)\}}{2\ell}$, where d is the Hamming-distance function. Intuitively, the larger the reconstruction rate is, the better the solution is.

To compare the all-heterozygous and the general cases, we use the datasets of Geraci (2010) for those combinations (ℓ, c, e) with $\ell \in \{100, 350\}$, $c \in \{3, 5, 8, 10\}$ and $e \in \{0\%, 10\%\}$. It turns out that when $e = 0\%$, our exact program for both cases can find the correct solution. On the other hand, when $e = 10\%$, the solutions for the two cases found by our exact program look different (cf. Table 3). In particular, the average reconstruction rate achieved by our exact program for the general case is better than the best reconstruction rate reported by Geraci (2010), but the average reconstruction rate achieved by our exact program for the all-heterozygous case is worse. Moreover, the larger c is, the worse the average reconstruction rate achieved by our exact program for the all-heterozygous case is. Consider a homozygous site p. For a fixed error rate e, with the increase of coverage at p, the chance that errors exist for reads at site p increases and so does the chance that p corresponds to a non-monotone column. Thus, the number of columns corresponding to homozygous sites in X increases. Therefore, the larger c is, the worse the average reconstruction rate achieved by the exact program for the all-heterozygous case is. This is the reason why we have to consider the general case. Contrary to the all-heterozygous case, the general case can handle such 'false-heterozygous' sites correctly, and thus the reconstruction rate increases with more coverage for the general case.

The running time of our program is also related to the coverage. With the increase of coverage, the number of errors increases and so does the MEC score. As the result, the running time increases accordingly as shown in Table 3.

To see how good our heuristics for the general case are, we use the datasets of Geraci (2010) for those combinations

Table 1. The running times of our program (HapCUT, respectively) and the optimal MEC scores (heuristic MEC scores, respectively) found for the HuRef dataset, where column 'chr' shows the index number of a chromosome, column 'score' shows the MEC score and column 'time' shows the running time

Chr	Our program				HapCUT	
	All-hete. case		General case			
	Score	Time	Score	Time	Score	Time
1	19 665	73	16 853	2872	19 762	186
2	14 647	10	12 618	21	14 680	177
3	10 688	8	9296	16	10 705	182
4	11 537	9	9958	19	11 566	188
5	10 558	9	9195	18	10 586	164
6	9884	8	8637	15	9913	158
7	11 246	8	9782	17	11 273	147
8	9800	12	8480	66	9829	151
9	9264	6	8051	13	9283	131
10	9815	9	8550	20	9849	154
11	8179	7	7027	14	8204	149
12	8213	7	7136	12	8238	133
13	5811	5	5090	10	5836	114
14	5844	4	5086	8	5851	85
15	9310	1573	8067∼8088	2788	9336	84
16	8235	9	7176	25	8269	89
17	6535	4	5739	8	6562	78
18	5019	4	4403	9	5033	90
19	5311	4	4628	8	5340	69
20	3741	3	3243	5	3753	59
21	3896	4	3360	40	3911	54
22	4507	70	3908	11 942	4539	37

Note that for some of the 22 chromosomes, our optimal MEC scores are larger than those of He *et al.* (2010). This is because He *et al.* (2010) exclude multi-gapped reads while we include all.

Table 2. The second heuristic results for the hard reduced-blocks of the HuRef dataset in the general case

Chr	Pos	#SNPs	Optimal		Heuristic	
			Score	Time	score	Time
1	11 500	461	789	2011	797	8
1	77 502	1015	1642	829	1644	52
8	14 931	1073	570	49	574	4
15	1	779	684	658	693	3
15	782	1166	?	?	2037	1566
21	76	354	536	35	536	2
22	1	398	1310	11 942	1310	68

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

Table 3. Evaluating our exact program using the simulated dataset of Geraci (2010) for those combinations (ℓ, c, e) with $\ell \in \{100, 350\}$, $c \in \{3, 5, 8, 10\}$ and $e = 10\%$, where column 'prev RR' shows the best average reconstruction rate reported by Geraci (2010) and columns 'org score', 'score', 'time' and 'RR' show the average MEC score of the correct solution, the average MEC score of the solution found by our exact program, the average time (in minutes) taken by our exact program, and the average reconstruction rate of our program over the 100 instances in the dataset for a particular combination $(\ell, c, 10\%)$, respectively

c	Prev	Org	General case			All-hete. case		
	RR	Score	Score	Time	RR	Score	Time	RR
$\ell = 100$								
3	93.0	59.1	53.3	0.03	96.2	66.6	0.02	92.7
5	98.5	97.3	95.4	0.16	99.0	132.9	0.11	92.0
8	99.3	157.7	157.3	0.62	99.7	249.4	0.52	90.1
10	99.8	196.0	195.8	0.94	99.9	327.2	1.02	89.2
$\ell = 350$								
3	93.0	207.0	186.2	0.45	96.5	233.5	0.31	93.0
5	97.8	345.6	338.4	3.05	98.8	477.2	2.68	91.4
8	99.6	548.3	547.1	16.5	99.7	909.8	41.4	88.6
10	99.8	682.0	682.0	30.5	99.9			

(ℓ, c, e) with $\ell \in \{100, 350\}$, $c \in \{3, 5, 8, 10\}$ and $e = 10\%$. For simplicity, we always choose the middle position as the cut position when experimenting with the first heuristic. The experimental results are summarized in Table 4. From the table, we can see that for the tested dataset, the average reconstruction rate achieved by our first (second, respectively) heuristic for the general case is better than the best reconstruction rate reported by Geraci (2010) when $c \geq 8$ ($c \leq 5$, respectively). It also turns out that for the tested dataset, the first heuristic is faster than the second. Furthermore, for those instances with $c \geq 5$ in the tested dataset, our second heuristic always finds optimal solutions, but its average reconstruction rate is different from that of our exact program for the general case. Consequently, the optimal solutions found by our second heuristic can be different from the optimal solutions found by our exact program for the general case.

3.3 The NA12878 dataset

This real dataset contains the whole-genome fosmid sequence data for an individual NA12878 (Duitama *et al.*, 2012). It is obtained by using fosmid pool-based next generation sequencing, which allows genome-wide generation of haploid DNA segments significantly larger than other standard shotgun sequencing technologies. We use the NA12878 dataset [to which some variant calling method has been applied by Duitama *et al.* (2011)] to compare our approach with the MaxSAT approach of He *et al.* (2010) and HapCUT (Bansal and Bafna, 2008). Our experimental results are summarized in Supplementary Table S11 from which one can see that our exact program can finish within 3 min for each of the 22 chromosomes in the dataset and is much faster than He *et al.* (2010)'s program. Besides, we can see that the optimal costs in the general case are smaller than those in the all-heterozygous case for all the 22 chromosomes in Supplementary Table S11. This implies that there are still some false-heterozygous columns in the filtered matrices.

Table 4. Evaluating our heuristics for the general case using the simulated dataset of Geraci (2010) for those combinations (ℓ, c, e) with $\ell \in \{100, 350\}$, $c \in \{3, 5, 8, 10\}$ and $e = 10\%$, where the columns mean the same as in Table 3

c	Prev	Org	First heuristic			Second heuristic		
	RR	Score	Score	Time	RR	Score	Time	RR
$\ell = 100$								
3	93.0	59.1	61.9	0.02	87.7	53.4	0.04	99.0
5	98.5	97.3	96.8	0.04	98.1	95.4	0.12	99.0
8	99.3	157.7	158.3	0.11	99.6	157.3	0.20	98.8
10	99.8	196.0	195.8	0.14	99.9	195.8	0.38	98.8
$\ell = 350$								
3	93.0	207.0	195.1	0.09	94.3	186.3	0.23	99.6
5	97.8	345.6	340.2	0.36	98.5	338.4	0.62	99.6
8	99.6	548.3	547.1	1.40	99.7	547.1	31.5	99.6
10	99.8	682.0	682.1	2.43	99.9			

3.4 Discussion

Multiple optimal solutions may exist for both the all-heterozygous and the general cases. If there is no error in the reads, the optimal solution is unique. If the error rate in reads is low, the chance that the optimal solution is unique is high. However, if the error rate is high, many optimal solutions may exist. In general, enumerating all optimal solutions are much harder and takes much longer time than computing a single optimal solution. From the experiments on simulated datasets, we can see that the single optimal solution found by our exact algorithm can achieve better reconstruction rate than the previously known heuristics in most cases. Still, it remains an open problem to handle multiple optimal solutions.

ACKNOWLEDGEMENTS

L.W. is supported by a grant from City University of Hong Kong [Project No. 7002728].

Conflict of Interest: none declared.

REFERENCES

Aguiar,D. and Istrail,S. (2012) HapCompass: a fast cycle basis algorithm for accurate haplotype assembly of sequence data. *J. Comput. Biol.*, **19**, 577–590.

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

Bansal,V. *et al.* (2008) An MCMC algorithm for haplotype assembly from whole genome sequence data. *Genome Res.*, **18**, 1336.

Beckmann,L. (2010) Haplotype Sharing Methods. In: *Encyclopedia of Life Sciences (ELS)*. John Wiley & Sons, Ltd, Chichester.

Chen,Z. *et al.* (2008) Linear time probabilistic algorithms for the singular haplotype reconstruction problem from SNP fragments. *J. Comput. Biol.*, **15**, 535–546.

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016

Cilibrasi,R. *et al.* (2005) On the complexity of several haplotyping problems. *Algorithms in Bioinformatics Lecture Notes in Computer Science*, **3692**, 128–139.
Clark,A. *et al.* (1998) Haplotype structure and population genetic inferences from nucleotide-sequence variation in human lipoprotein lipase. *Am. J. Hum. Genet.*, **63**, 595–612.
Deng,F. *et al.* (2013) A highly accurate heuristic algorithm for the haplotype assembly problem. *BMC Genomics*, **14**(**Suppl. 2**), S2.
Duitama,J. *et al.* (2010) ReFHap: A reliable and fast algorithm for single individual haplotyping. In *Proceedings of the First ACM International Conference on Bioinformatics and Computational Biology*, Niagara Falls, NY, USA, pp. 160–169.
Duitama,J. *et al.* (2012) Fosmid-based whole genome haplotyping of a HapMap trio child: evaluation of Single Individual Haplotyping techniques. *Nucleic Acids Res.*, **40**, 2041–2053.
Duitama,J. *et al.* (2011) Towards accurate detection and genotyping of expressed variants from whole transcriptome sequencing data. In *Proceedings of ICCABS*, Orlando, FL, USA, pp. 87–92.
Eskin,E. *et al.* (2003) Efficient reconstruction of haplotype structure via perfect phylogeny. *J. Bioinform. Comput. Biol.*, **1**, 1–20.
Geraci,F. (2010) A comparison of several algorithms for the single individual SNP haplotyping reconstruction problem. *Bioinformatics*, **26**, 2217–2225.
Halperin,E. and Eskin,E. (2004) Haplotype reconstruction from genotype data using imperfect phylogeny. *Bioinformatics*, **20**, 1842–1849.
He,D. *et al.* (2010) Optimal algorithms for haplotype assembly from whole-genome sequence data. *Bioinformatics*, **26**, i183–i190.
Hoehe,M. *et al.* (2000) Sequence variability and candidate gene analysis in complex disease: association of μ opioid receptor gene variation with substance dependence. *Hum. Mol. Genet.*, **9**, 2895–2908.
Lancia,G. *et al.* (2001) SNPs problems, complexity, and algorithms. In *Proceedings of the 9th Annual European Symposium on Algorithms*, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, **2161**, pp. 182–193.
Levy,S. *et al.* (2007) The diploid genome sequence of an individual human. *PLoS Biol.*, **5**, e254.
Li,L.M. *et al.* (2004) Haplotype reconstruction from SNP alignment. *J. Comput. Biol.*, **11**, 505–516.
Lippert,R. *et al.* (2002) Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Brief. Bioinform.*, **3**, 23–31.
Panconesi,A. and Sozio,M. (2004) Fast Hare: a fast heuristic for single individual SNP haplotype reconstruction. *Algorithms in Bioinformatics Lecture Notes in Computer Science*, **3240**, 266–277.
Schwartz,R. *et al.* (2002) Methods for inferring block-wise ancestral history from haploid sequences. *Algorithms in Bioinformatics Lecture Notes in Computer Science*, **2452**, 44–59.
Schwartz,R. (2010) Theory and algorithms for the haplotype assembly problem. *Commun. Inform. Syst.*, **10**, 23–38.
Stephens,M. *et al.* (2001) A new statistical method for haplotype reconstruction from population data. *Am. J. Hum. Genet.*, **68**, 978–989.
Wang,J. *et al.* (2010) A practical exact algorithm for the individual haplotyping problem MEC/GI. *Algorithmica*, **56**, 283–296.
Wang,L. and Xu,Y. (2003) Haplotype inference by maximum parsimony. *Bioinformatics*, **19**, 1773–1780.
Wang,R.S. *et al.* (2005) Haplotype reconstruction from SNP fragments by minimum error correction. *Bioinformatics*, **21**, 2456–2462.
Xie,M. *et al.* (2012) A fast and accurate algorithm for single individual haplotyping. *BMC Syst. Biol.*, **6** (**Suppl. 2**), S8.
Wu,L.Y. *et al.* (2009) Self-organizing map approaches for the haplotype assembly problem. *Math. Comput. Simul.*, **79**, 3026–3037.
Zhao,Y.Y. *et al.* (2005) Haplotype assembly from aligned weighted SNP fragments. *Comput. Biol. Chem.*, **29**, 281–287.

Downloaded from http://bioinformatics.oxfordjournals.org/ at :: on August 30, 2016