Bio-Medical Materials and Engineering 24 (2014) 3753–3762
DOI 10.3233/BME-141204
IOS Press

Haplotyping a single triploid individual based on genetic algorithm

Jingli Wu*, Xixi Chen and Xianchen Li
Department of Computer Science, Guangxi Normal University, Guilin 541004, China

Abstract. The minimum error correction model is an important combinatorial model for haplotyping a single individual. In this article, triploid individual haplotype reconstruction problem is studied by using the model. A genetic algorithm based method GTIHR is presented for reconstructing the triploid individual haplotype. A novel coding method and an effectual hill-climbing operator are introduced for the GTIHR algorithm. This relatively short chromosome code can lead to a smaller solution space, which plays a positive role in speeding up the convergence process. The hill-climbing operator ensures algorithm GTIHR converge at a good solution quickly, and prevents premature convergence simultaneously. The experimental results prove that algorithm GTIHR can be implemented efficiently, and can get higher reconstruction rate than previous algorithms.

Keywords: Triploid, haplotype, Single nucleotide polymorphism (SNP), the minimum error correction, genetic algorithm

1. Introduction

Triploid genomes are widespread in nature, especially in the genomes of fish, shellfish and higher plants. It is well known that triploid fish and shellfish have some excellent characteristics such as fast growth, high survival rate and strong disease resistance. Triploid plants also have such advantages as luxuriant growth, bigger fruits and strong adaptability. Therefore, it is very imperative to investigate the genetic characteristics of triploid genomes, for improving the quality and yield of triploid species. It is widely accepted that haplotypes are the predominant form to address phenotypic difference and disease susceptibility, but it is technically difficult and expensive to obtain haplotypes via biological experiments directly. Therefore, computation has become a general way to obtain haplotypes, and the individual haplotype reconstruction problem has arisen widely attention in bioinformatics.

At present, although many reconstruction algorithms have been proposed for haplotyping a diploid individual [1–3], there is relatively little research on reconstructing triploid ones. Some reconstruction methods were presented for solving the *K*-individual haplotype reconstruction problem. Wang et al. proposed a genetic algorithm (it is named as W_GA in this article) to assemble diploid individual haplotypes, which can be adapted to the *K*-individual haplotype reconstruction problem [4]. Li et al. [5] presented an exact algorithm for solving this problem. Qian et al. proposed a particle swarm optimization algorithm (it is named as Q_PSO in this article) for solving diploid individual haplotypes reconstruction, which can also be adapted to the *K*-individual haplotype reconstruction problem [6]. The

*Corresponding author: Jingli Wu, Department of Computer Science, Guangxi Normal University, Guilin 541004, China. Tel.: +86 13617832552; Fax: +86-773-5811621; E-mail: wjlhappy@mailbox.gxnu.edu.cn.

above mentioned algorithms can be used to reconstruct triploid individual haplotype (i.e., K is assigned to 3), but they are all inefficient in practical applications. The code length of algorithms W_GA and Q_PSO equals the amount of SNP fragments, which is very huge in realistic applications. The reconstruction rates obtained by these algorithms are not high due to their long codes [7]. With the increase of chromosome ploidy, the number of SNP fragments increases. Hence, the reconstruction rates got by these algorithms will decrease further. The dynamic programming algorithm presented by Li et al. [5] performs well when the amount of fragments is small, but does not scale well with the increase of fragment number. In this article, triploid individual haplotype reconstruction problem is studied based on the minimum error correction model. By devising a novel chromosome coding method and an effectual hill-climbing operator, algorithm GTIHR is introduced for solving this model.

The article is organized as follows. In Section 2, definitions and notations are given. In Section 3, algorithm GTIHR is depicted. In Section 4, the experimental results are presented. Finally, several conclusions are reached in Section 5.

2. Definitions and notations

A triploid individual owns three haplotypes coming from three chromosomes for a given sequence of SNPs. Because almost all common SNPs have merely two alleles, a haplotype can be denoted by a binary string. A SNP site is called homozygous if it possesses the same allele on the three haplotypes, otherwise it is called heterozygous. Assume that m fragments come from three chromosomes, and the haplotypes length is n. Let $M_{m\times n}$ denote an $m\times n$ SNP matrix, where each entry $M[i,j](i=1,2,\ldots,m, j=1,2,\ldots,n)$ equals 0, 1 or – (– denotes the value is null). Each row denotes a fragment and each column denotes an SNP site. Let $n_x(j)$ ($j=1,2,\ldots,n$) record the amount of x entries in the j-th column. Let $f_x(j)$ store the proportion of x entries in the non-null entries of column j, i.e., $f_x(j)=n_x(j)/(n_x(j)+n_{1-x}(j))$. Given $u, v\in\{0,1,-\}$, $s(u, v)$ and $d(u, v)$ are defined as Eqs. (1) and (2).

$$s(u,v)=\begin{cases}1, & \text{if } u\neq -, v\neq -, \text{and } u=v,\\ 0, & \text{otherwise,}\end{cases} \tag{1}$$

$$d(u,v)=\begin{cases}1, & \text{if } u\neq -, v\neq -, \text{and } u\neq v,\\ 0, & \text{otherwise}\end{cases} \tag{2}$$

Given two strings $U=<u_1u_2\ldots u_n>$ and $V=<v_1v_2\ldots v_n>$, where $u_i, v_i\in\{0,1,-\}(i=1,2,\ldots,n)$, $D(U,V)$ records the number of sites having opposite values in strings U and V, and is defined in Eq. (3):

$$D(U,V)=\sum_{i=1}^{n} d(u_i,v_i). \tag{3}$$

Assume that U and V are two fragments, they are called *conflict* if $D(U,V)$ is larger than 0, and are called *agree* if $D(U,V)$ equals 0. The conflict between two fragments indicates that they are probably sequenced from different chromosome copies or have sequencing errors. If all of the fragments have no sequencing errors, the rows in matrix M can be divided into three groups of non-conflict fragments, which can be assembled into three haplotypes. In this case, the matrix M is called *error-free*.

In this article, the *Minimum Error Correction* model [8] is used to haplotyping a triploid individual, as follows: Given an SNP matrix M, correct the minimal amount of elements to make M *error-free*.

Assume that the rows $M[i, -](i=1,2,\ldots,m)$ in matrix M are divided into three mutually exclusive groups, and matrix G denotes any one of the three groups, i.e., $G[l,-]=M[i_l,-]$ ($l=1,2,\ldots,k$, $1\le i_k, k\le m$). Let $N_0(G,j)$ (resp. $N_1(G,j)$) denote the amount of 0 (resp. 1) entries in the group $\{G[1,j], G[2,j], \ldots, G[k,j]\}$, and string $h(G)=<h_1(G)h_2(G)\ldots h_n(G)>$ denote the assembled haplotypes of G, where

$$h_j(G)=\begin{cases}0, \text{ if } N_0(G,j)>N_1(G,j), j=1,2,\ldots,n,\\ 1, \text{ otherwise},\end{cases} \tag{4}$$

The reconstruction rate (*RR*) [2,4] is defined as the proportion of SNPs that are assembled correctly. Suppose that $h_o=(h_{o1}, h_{o2}, h_{o3})$ are the original haplotypes, and $h_a=(h_{a1}, h_{a2}, h_{a3})$ are the assembled ones. Let $r_{ij}=D(h_{oi},h_{aj})$ $(i,j=1, 2, 3)$. *RR* is computed as Eq. (5):

$$RR(h_o,h_a)=1-\frac{\min\left\{\sum_{k=1}^{3} r_{i_k j_k}\middle| i_k, j_k\in\{1,2,3\}, \sum_{k=1}^{3} i_k=\sum_{k=1}^{3} j_k=6\right\}}{3n}, \tag{5}$$

3. Reconstruction method GTIHR

In this section, a genetic algorithm based method GTIHR is described. The input consists of an SNP matrix $M_{m\times n}$ and a parameter τ. The output is three assembled n-length haplotypes $h=(h_1, h_2, h_3)$. Firstly, the matrix M is preprocessed. Secondly, genetic algorithm is executed to generate haplotypes $\hat{h}=(\hat{h}_1,\hat{h}_2,\hat{h}_3)$ having merely heterozygous sites. Finally, the resulting haplotypes $h=(h_1, h_2, h_3)$ are got by augmenting $\hat{h}$. The following are key steps in algorithm GTIHR.

3.1. Preprocessing

In the preprocessing stage, some unnecessary information is removed. If the j-th ($j=1,2,\ldots,n$) column meets the condition of $f_0(j)\le\tau$ (resp. $f_1(j)\le\tau$), it is dropped and is labeled as 1-column (resp. 0-column), here τ is set to 0.2 [9]. After dropping columns, all of the retained SNP sites are heterozygous, and the rows with all – elements are also discarded, for they are useless in the reconstruction process. The preprocessed matrix is still represented by $M_{m\times n}$.

3.2. Genetic algorithm

Genetic algorithm is a kind of metaheuristic algorithm for solving complex problems, and it has been used to solve successfully many combinatorial optimization problems in bioinformatics [10,11]. Some interrelated key techniques in devising the genetic algorithm are given as follows.

3.2.1. Coding method

A two-tuple (X,Y) is used to represent a chromosome, where $X=<x_1x_2\ldots x_n>$ ($x_j\in\{0,1\}$, $j=1,2,...,n$) and $Y=<y_1y_2\ldots y_n>$ ($y_j\in\{0,1,-1\}$, $j=1,2,...,n$). X denotes a haplotype with only heterozygous SNPs, i.e., $\hat{h}_1$. Y is used to indicate the relationship of the alleles among haplotypes $\hat{h}_1$, $\hat{h}_2$ and $\hat{h}_3$ that have only heterozygous SNPs, as shown in Eqs. (6) and (7). Three haplotypes, which have only heterozygous SNPs, can be computed from a chromosome (X,Y).

$$\hat{h}_{2j}=\begin{cases}1-x_j, & \text{if } y_j\in\{0,-1\},\\ x_j, & \text{if } y_j=1,\end{cases} \tag{6}$$

$$\hat{h}_{3j}=\begin{cases}1-x_j, & \text{if } y_j\in\{0,1\},\\ x_j, & \text{if } y_j=-1,\end{cases} \tag{7}$$

3.2.2. The initial population

Let $(X_i(T),Y_i(T))$ ($i=1,2,...,N$, N denotes the population size) denote the i-th chromosome of the T-th generation. The i-th initial chromosome $(X_i(0), Y_i(0))$ is created with the following method: divide the rows of matrix M into three sets randomly, and three n-length haplotypes $(\hat{h}_1,\hat{h}_2,\hat{h}_3)$ with only heterozygous SNPs are generated according to Eq. (4). Then choose $\hat{h}_1$ to be chromosome $X_i(0)$. $Y_i(0)=<y_{i1}(0)y_{i2}(0)\ldots y_{in}(0)>$, where $y_{ij}(0)$ ($j=1,2,\ldots,n$) is computed using Eq. (8). The initial population is created by generating N initial chromosomes.

$$y_{ij}(0)=\begin{cases}0, & \text{if } \hat{h}_{2j}=\hat{h}_{3j},\ \hat{h}_{2j}\neq\hat{h}_{1j},\ \hat{h}_{3j}\neq\hat{h}_{1j},\\ 1, & \text{if } \hat{h}_{2j}\neq\hat{h}_{3j},\ \hat{h}_{2j}=\hat{h}_{1j},\ \hat{h}_{3j}\neq\hat{h}_{1j},\\ -1, & \text{if } \hat{h}_{2j}\neq\hat{h}_{3j},\ \hat{h}_{2j}\neq\hat{h}_{1j},\ \hat{h}_{3j}=\hat{h}_{1j}.\end{cases} \tag{8}$$

3.2.3. Operator

In the GTIHR algorithm, the roulette wheel selection mechanism is adopted to create next-generation population. The combination of uniform crossover and one-point crossover is used, i.e., both of them are used with the probability of 50%. Similarly, the combination of swap mutation and one-point mutation is used. Hill-climbing operator can be used to generate new search regions. In this paper, a novel hill-climbing operator is devised. The following steps depict the operator:

1) Given a chromosome (X,Y), a random binary string *mask* of length n is generated. When the j-th ($j=1,2,\ldots,n$) bit of *mask* is equal to ''0'', x_j is changed to 1-x_j, and y_j is changed to any element of set $\{0,1,-1\}$-$\{y_j\}$ uniformly, otherwise x_j and y_j are both unchanged.

2) Divide all of the fragments $M[i,-]$($i=1,2,\ldots,m$) in matrix M into three sets according to (X,Y). Given $M[i,-]$, let $S_k(X,Y,M[i,-])$($k=1,2,3$) record the amount of SNP sites that have the same allele in $M[i,-]$ and $\hat{h}_k$, as Eqs. (9)-(11). $M[i,-]$ is classified into group G_K, $K=\underset{k=1,2,3}{\operatorname{argmax}}\{S_k(X,Y,M[i,-])\}$.

$$S_1(X,Y,M[i,-]) = \sum_{j=1}^{n} s(x_j, M[i,j]), \tag{9}$$

$$S_2(X,Y,M[i,-]) = \sum_{j=1}^{n} (s(x_j, M[i,j]) \cdot (y_j = 1) + d(x_j, M[i,j]) \cdot (y_j \in \{0,-1\})), \tag{10}$$

$$S_3(X,Y,M[i,-]) = \sum_{j=1}^{n} (s(x_j, M[i,j]) \cdot (y_j = -1) + d(x_j, M[i,j]) \cdot (y_j \in \{0,1\})). \tag{11}$$

As shown in Eqs. (6) and (7), because there exists correspondence relation between chromosome (X,Y) and haplotypes $\hat{h}_2$, $\hat{h}_3$, $S_2(X,Y,M[i,-])$ and $S_3(X,Y,M[i,-])$ need to be computed according to each site value of string Y. Take Eq. (10) for example, if $y_j \in \{0,-1\}$, $\hat{h}_{2j}$ has different value from x_j, hence $d(x_j, M[i,j])$ can be used to substitute $s(\hat{h}_{2j}, M[i,j])$.

3) Compute six arrays $dif[k]$, $same[k]$(k=1,2,3) as follows:

$$dif[1][j] = \sum_{M[i,-] \in G_1} d(x_j, M[i,j]),\ i = 1,2,\ldots,m,\ j = 1,2,\ldots,n, \tag{12}$$

$$same[1][j] = \sum_{M[i,-] \in G_1} s(x_j, M[i,j]),\ i = 1,2,\ldots,m,\ j = 1,2,\ldots,n, \tag{13}$$

$$dif[2][j] = \sum_{M[i,-] \in G_2} (d(x_j, M[i,j]) \cdot (y_j = 1) + s(x_j, M[i,j]) \cdot (y_j \in \{0,-1\})), i = 1,2,\ldots,m,\ j = 1,2,\ldots,n, \tag{14}$$

$$same[2][j] = \sum_{M[i,-] \in G_2} (s(x_j, M[i,j]) \cdot (y_j = 1) + d(x_j, M[i,j]) \cdot (y_j \in \{0,-1\})), i = 1,2,\ldots,m,\ j = 1,2,\ldots,n, \tag{15}$$

$$dif[3][j] = \sum_{M[i,-] \in G_3} (d(x_j, M[i,j]) \cdot (y_j = -1) + s(x_j, M[i,j]) \cdot (y_j \in \{0,1\})), i = 1,2,\ldots,m,\ j = 1,2,\ldots,n, \tag{16}$$

$$same[3][j] = \sum_{M[i,-] \in G_3} (s(x_j, M[i,j]) \cdot (y_j = -1) + d(x_j, M[i,j]) \cdot (y_j \in \{0,1\})), i = 1,2,\ldots,m,\ j = 1,2,\ldots,n, \tag{17}$$

here $dif[k][j]$ counts the amount of fragments in set G_k which possess different alleles with $\hat{h}_k$ on the j-th site. $same[k][j]$ counts the amount of fragments in set G_k which possess the same allele with $\hat{h}_k$ on the j-th site.

4) If $\sum_{k=1}^{3} dif[k][j] > \sum_{k=1}^{3} same[k][j]$ (j=1,2,…,n), x_j is converted to 1-x_j, otherwise x_j remains unchanged.

3.2.4. Fitness function

Given a chromosome (X,Y), a fitness function $Fitness(X,Y)$ need to be designed to measure how ‘‘good’’ the chromosome, as defined in Eqs. (18)-(22):

$$Fitness\,(X,Y)=1-\frac{E(X,Y)}{m\cdot n}, \tag{18}$$

$$E(X,Y)=\sum\nolimits_{1\le i\le m}\min\nolimits_{k=1,2,3}(D_k(X,Y,M[i,-])), \tag{19}$$

$$D_1(X,Y,M[i,-])=\sum\nolimits_{j=1}^{n} d(x_j,M[i,j]) \tag{20}$$

$$D_2(X,Y,M[i,-])=\sum\nolimits_{j=1}^{n}(d(x_j,M[i,j])\cdot(y_j=1)+s(x_j,M[i,j])\cdot(y_j\in\{0,-1\})), \tag{21}$$

$$D_3(X,Y,M[i,-])=\sum\nolimits_{j=1}^{n}(d(x_j,M[i,j])\cdot(y_j=-1)+s(x_j,M[i,j])\cdot(y_j\in\{0,1\}))\text{..} \tag{22}$$

Here $D_k(X,Y,M[i,-])$ (k=1,2,3) counts the amount of sites which possess different alleles on $M[i,-]$ and $\hat{h}_k$. $E(X,Y)$ denotes the minimum amount of error corrections corresponding to the three haplotypes $(\hat{h}_1,\hat{h}_2,\hat{h}_3)$. Therefore, the smaller $E(X,Y)$ is, the higher the fitness of chromosome (X,Y) is. The presented genetic algorithm is summarized in Figure 1.

3.3. Augmenting

The homozygous SNPs, which are removed at the preprocessing stage, should be reinserted. Given haplotypes $\hat{h}=(\hat{h}_1,\hat{h}_2,\hat{h}_3)$, $\hat{h}_1$, $\hat{h}_2$ and $\hat{h}_3$ are augmented by the SNP sites discarded in the preprocessing stage to generate h_1, h_2 and h_3. If a discarded column j is 0-column (resp.1- column), position j of the three haplotypes will also be 0 (resp.1). Then the resulting haplotypes h=(h_1, h_2, h_3) are obtained.

The description of the GTIHR algorithm is omitted here because of space limitations.

Algorithm 1 Genetic algorithm
Input: the preprocessed matrix M, population size N, crossover probability p_c, mutation probability p_m, the maximum number of population generation *maxgen*
Output: three haplotypes $\hat{h}=(\hat{h}_1,\hat{h}_2,\hat{h}_3)$
Step1. Generate the initial population pop_0, *gen*=0.
Step2. If *gen*>*maxgen*, go to **Step7**, otherwise go to **Step3**.
Step3. Select (1-p_c)×N members from pop_{gen} by using selection operator, and add them to pop_{gen+1}.
Step4. Select p_c×N/2 pairs of members from pop_{gen} by using selection operator. For each pair, produce two offspring by randomly applying single-point crossover operator and uniform crossover operator. Add all the offspring to pop_{gen+1}.
Step5. Select p_m×N individuals from pop_{gen+1} with uniform probability. For each individual, produce an offspring by randomly applying single-point mutation operator and swap mutation operator.
Step6. For each individual i (i=1,2,…,N) in pop_{gen+1}, produce a new one i' by using hill-climbing operator. If the fitness of individual i' is larger than that of individual i, replace i with i', otherwise i remains unchanged. *gen*=*gen*+1, go to **Step2**.
Step7. Transform the individual with the highest fitness into three haplotypes $\hat{h}=(\hat{h}_1,\hat{h}_2,\hat{h}_3)$, Output $\hat{h}$.

Fig. 1. Genetic algorithm.

4. Experiment results

In this section, simulation data were generated under some realistic assumptions. Three n-length haplotypes h=(h_1, h_2, h_3) were created as follows: h_1 was generated at random; h_2 was generated by flipping each bit of h_1 at random so that the hamming distance between h_1 and h_2 equaled a parameter d; h_3 also had the same length and each bit of which was assigned the corresponding bit of h_1 or h_2 randomly, i.e., h_{3j} was assigned h_{1j} or h_{2j} (j=1,2,…,n) at random. With regard to fragment data, as far as we know, real DNA fragments data are not available in the public domain. Therefore, in the experiments, an extensively used sequence simulator CELSIM [7,9,12] was adopted to produce simulated fragments. m_1 single SNP fragments and m_2 mate-pair SNP fragments were produced. A single fragment length was between *f_min* and *f_max*, and a mate-pair fragment length was set to n/10. In realistic applications, by using Sanger sequencing technology, a single fragment length ranges between 3 and 7 [9]. With the advance of next generation sequencing (NGS) technologies, e.g., 454 sequencing [13], a single SNP fragment length is shortened. Hence, the algorithms were also tested with shorter single SNP fragment length. Both single and mate-pair fragments have the coverage of c/2, and the total coverage was c. Finally, reading errors were planted into the fragments with probability p_s. In realistic applications, c is between 5 and 10, and p_s is between 2% and 5% [9].

Reconstruction rate and execution time are used to evaluate algorithms GTIHR, W_GA and Q_PSO. For each setting of parameter values, 100 datasets were produced, and the average over 100 runs was computed and presented. The parameters of algorithms W_GA and Q_PSO were set according to reference [4] and [6], respectively. The parameters of algorithm GTIHR were set as follows: p_c=0.8, p_m=0.2, N=400, *maxgen*=150, which are the same as the W_GA algorithm. The algorithms were implemented on a Windows Server (Intel Core i3-2100 3.10 GHz CPU and 4 GByte RAM) using Microsoft Visual C++ compiler 6.

Table 1

Performance comparisons with different p_s values

p_s	RR			Execution time(s)		
	GTIHR	W-GA	Q-PSO	GTIHR	W-GA	Q-PSO
0.01	0.99	0.93	0.94	22.69	19.64	14.98
0.02	0.98	0.92	0.92	22.45	19.16	14.32
0.03	0.98	0.92	0.92	20.16	19.88	14.86
0.04	0.98	0.91	0.91	22.81	19.91	15.03
0.05	0.98	0.91	0.90	21.66	19.64	14.75
0.1	0.94	0.87	0.84	24.17	19.14	14.97

Table 2

Performances comparisons with different c values

c	RR			Execution time (s)		
	GTIHR	W-GA	Q-PSO	GTIHR	W-GA	Q-PSO
2	0.89	0.83	0.83	5.70	4.53	2.52
4	0.91	0.87	0.87	9.06	7.90	5.61
6	0.92	0.89	0.89	11.98	10.66	8.07
8	0.94	0.90	0.90	17.03	16.36	11.32
10	0.98	0.91	0.90	21.66	19.64	14.75

Table 3

Performances comparisons with different n values

n	*RR*			Execution time(s)		
	GTIHR	W-GA	Q-PSO	GTIHR	W-GA	Q-PSO
100	0.98	0.91	0.903	21.66	19.64	14.75
200	0.923	0.875	0.866	52.94	50.08	22.11
500	0.893	0.869	0.874	294.45	267.59	90.71
800	0.881	0.868	0.870	689.63	639.58	206.43
1000	0.872	0.864	0.865	1156.55	977.98	320.28

In Table 1, the value of error rate p_s ranges from 0.01 to 0.1, and *f_min*=3, *f_max*=7, *c*=10, *n*=100, *d*=0.3. It can be seen from Table 1, with the increase of error rate, the *RR*s of the three algorithms decrease. The GTIHR algorithm has a higher *RR* than the other two algorithms at each setting of p_s. When p_s increases from 0.01 to 0.1, the *RR* of the GTIHR algorithm decreases from 0.99 to 0.94, the *RR* of the W_GA algorithm decreases from 0.93 to 0.87, and the *RR* of the Q_PSO algorithm decreases from 0.94 to 0.84. Additionally, the three algorithms run quickly, and the execution time of them is not affected by parameter p_s.

Table 2 represents the experiment results of the three algorithms under different coverage *c*, where *f_min*=3, *f_max*=7, *n*=100, p_s=0.05, *d*=0.3. The results indicate that the *RR*s of the three algorithms increase with the augment of *c*, for more original fragment information can be offered to the algorithms. Algorithm GTIHR can obtain higher *RR* than algorithms W_GA and Q_PSO at each setting of *c*. The *RR* of the GTIHR algorithm increases from 0.89 to 0.98, the *RR* of the W_GA algorithm increases from 0.83 to 0.91, and the *RR* of the Q_PSO algorithm increases from 0.83 to 0.90. The three algorithms behave similarly in execution efficiency.

Table 3 compares the three algorithms with different haplotype length *n*, where *f_min*=3, *f_max*=7, *c*=10, p_s=0.05, *d*=0.3. As represented in Table 3, with the increase of *n*, the *RR*s of the three algorithms decrease, and the *RR*s of algorithm GTIHR are higher than those of algorithms W_GA and Q_PSO at each setting of *n*. The execution time of the three algorithms is very sensitive to *n*.

Table 4 represents the experiment results under different single fragment length ranges, and *c*=10, p_s=0.05, *n*=100, *d*=0.3. The decrease of the single fragment length decreases the probability of fragments overlapping, which does not contribute to reconstructing haplotypes, the experimental results clearly supported this idea. From Table 4, it can be seen that the GTIHR algorithm can obtain higher RR than the W_GA and Q_PSO algorithms at each setting of [*f_min*, *f_max*]. The execution time of algorithm GTIHR has lower sensitivity to parameter [*f_min*, *f_max*] variations compared to those of algorithms W_GA and Q_PSO. When [*f_min*, *f_max*] changes from [3,7] to [1,2], the execution time of algorithms GTIHR, W_GA and Q_PSO increase by about 0.25, 2.56 and 0.73 times respectively.

Table 4

Performance comparisons with different single fragment length ranges

[*f*_min,*f*_max]	RR			Execution time(s)		
	GTIHR	W-GA	Q-PSO	GTIHR	W-GA	Q-PSO
[3,7]	0.98	0.91	0.90	21.66	19.64	14.75
[2,4]	0.93	0.89	0.80	23.68	26.74	16.37
[1,2]	0.91	0.87	0.87	27.11	69.96	25.48

Table 5

Performances comparisons with different *d* values

d	RR			Execution time(s)		
	GTIHR	W-GA	Q-PSO	GTIHR	W-GA	Q-PSO
0.1	0.99	0.97	0.97	5.18	17.55	13.87
0.3	0.98	0.91	0.90	21.66	19.64	14.75
0.5	0.94	0.83	0.85	31.93	18.78	14.41
0.7	0.81	0.76	0.72	48.06	19.38	14.17
0.9	0.75	0.72	0.70	58.84	18.43	14.77
1	0.75	0.71	0.64	62.05	19.22	15.06

Table 5 shows the comparison results with different hamming distances *d*, and c=10, p_s=0.05, f_min=3, f_max=7, n=100. With the increase of *d*, the *RR*s of the three algorithms decrease gradually, and algorithm GTIHR can get higher *RR* than algorithms W_GA and Q_PSO at each setting of *d*. Assume that a haplotype has fixed length, the number of heterozygous sites in the haplotype increases with the augment of *d*. As can be seen from the chromosome code of algorithm GTIHR, the code length is twice the number of heterozygous sites. The larger *d* is, the longer the chromosome code is, and the solution space is enlarged. Therefore, when the population size and the maximum generation number are fixed, the increase of *d* decreases the *RR* of algorithm GTIHR, for large solution space plays a negative role in the performance of genetic algorithm. In addition, with the increase of *d*, although the execution time of algorithm GTIHR increases gradually, it still runs in high efficiency.

The performances of the three algorithms closely relate to search space size, which is determined directly by the code length. As mentioned above, the code length of algorithm GTIHR, which is twice the amount of heterozygous SNPs of a haplotype, is about $2nd$ bits. The codes of algorithms W_GA and Q_PSO have a length of $3nc/(f\_min+f\_max)+15c$ bits, which equals the amount of fragments. In a general case, the code of algorithm GTIHR is shorter than those of algorithms W_GA and Q_PSO. For instance, when f_min=3, f_max=7, n=100, c=10, both the W_GA and the Q_PSO algorithms have a code length of about 450, while the GTIHR algorithm has a code length between 20 and 200, which is determined by parameter *d*. Therefore, algorithm GTIHR has a much smaller search space than the other two algorithms, and can find a good solution more easily than them. In addition, the designed hill-climbing operator takes full advantage of the information provided by fragments to modify the chromosomes, and makes the chromosomes evolving towards higher fitness. Hence algorithm GTIHR can quickly converge to a satisfying solution. The operator still injects randomness to improve the hill-climbing ability of algorithm GTIHR, making it escape from the local optimum solution.

5. Conclusion

In this article, a genetic algorithm based method GTIHR is proposed for solving the triploid individual haplotype reconstruction problem. The GTIHR algorithm adopts a novel short chromosome code and an effectual hill-climbing operator, which has such advantages as restricting the size of the search space and enhancing hill climbing ability for algorithms. Comparing with algorithms W_GA and Q_PSO, algorithm GTIHR can achieve higher reconstruction rate haplotypes under different parameter settings, which was proven by an amount of experiments. Simultaneously, algorithm GTIHR is intended to rebuild haplotypes from a set of shorter fragments; it still keeps good performance with a decrease in the single fragment length. Therefore, the GTIHR algorithm may also adapt to the NGS

technologies. In summary, algorithm GTIHR is a practical solution for reconstructing a triploid single individual haplotypes.

Acknowledgement

The authors are grateful to Professor Gene Myers for his kindly providing CELSIM. This research is supported by the scientific research project of Guangxi Education Department under Grant No. 2013YB028, the National Natural Science Foundation of China under Grant No. 61363035, No. 61165009 and No. 61202272, "Bagui Scholar" Project Special Funds.

References

[1] G. Lancia, V. Bafna, S. Istrail, R. Lippert and R. Schwartz, SNPs problems, complexity and algorithms, in: Proceedings of the 9th Annual European Symposium on Algorithms, F.M. Heide, ed., Springer-Verlag, London, 2001, pp. 182–193.
[2] F. Geraci, A comparison of several algorithms for the single individual SNP haplotyping reconstruction problem, Bioinformatics **26** (2010), 2217–2225.
[3] F. Deng, W.J. Cui and L.S. Wang, A highly accurate heuristic algorithm for the haplotype assembly problem, BMC Genomics **14** (2013), S2.
[4] R.S. Wang, L.Y. Wu, Z.P. Li and X.S. Zhang, Haplotype reconstruction from SNP fragments by minimum error correction, Bioinformatics **21** (2005), 2456–2462.
[5] Z.P. Li, L.Y. Wu, Y.Y. Zhao and X.S. Zhang, A dynamic programming algorithm for the k-haplotyping problem, Acta. Mathematicae. Sinica. English Series **22** (2006), 405–412.
[6] W.Y. Qian, Y.J. Yang, N.N. Yang and L. Chun, Particle swarm optimization for snp haplotype reconstruction problem, Applied Mathematics and Computation **196** (2008), 266–272.
[7] J.L. Wu, J.X. Wang and J.E. Chen, A parthenogenetic algorithm for single individual SNP haplotyping, Engineering Applications of Artificial Intelligence **22** (2009), 401–406.
[8] R. Lippert, R. Schwartza, G. Lancia and S. Istrail, Algorithmic strategies for the SNPs haplotype assembly problem, Briefings in Bioinformatics **3** (2002), 23–31.
[9] A. Panconesi and M. Sozio, Fast hare: A fast heuristic for single individual SNP haplotype reconstruction, in: Proceedings of the 4th Int. Workshop on Algorithms in Bioinformatics, I. Jonassen and J. Kim, ed., Springer-Verlag, Berlin, 2004, pp. 266–277.
[10] S.N. Yu and M.Y. Lee, Bispectral analysis and genetic algorithm for congestive heart failure recognition based on heart rate variability, Computers in Biology and Medicine **42** (2012), 816–825.
[11] K. Anekboon, C. Lursinsap, S. Phimoltares, S. Fucharoen and S. Tongsima, Extracting predictive SNPs in Crohn's disease using a vacillating genetic algorithm and a neural classifier in case-control association studies, Computers in Biology and Medicine **44** (2014), 57–65.
[12] G. Myers, A dataset generator for whole genome shotgun sequencing, in: Proceedings of the 7th Int. Conf. on Intelligent Systems for Molecular Biology, T. Lengauer, R. Schneider, P. Bork et al., ed., AAAI, Heidelberg, 1999, pp. 202–210.
[13] M. Margulies, Genome sequencing in microfabricated high-density picolitre reactors, Nature **437** (2005), 376–380.