A Parthenogenetic Algorithm for the Founder Sequence Reconstruction Problem

Jingli Wu
College of Computer Science and Information Technology, Guangxi Normal University, Guilin, 541004, China
Email: wjlhappy@mailbox.gxnu.edu.cn

Hua Wang
College of Computer Science and Information Technology, Guangxi Normal University, Guilin, 541004, China
Email: hwang1986@126.com

***Abstract*—The maximum fragment length (MFL) is an important computational model for solving the founder sequence reconstruction problem. Benedettini et al. presented a meta-heuristic algorithm BACKFORTH based on iterative greedy method. The BACKFORTH algorithm starts with a single initial solution, and iteratively alternates between a partial destruction and reconstruction in order to obtain a final solution. The kind of optimization mechanism, which is based on a single initial solution, may make the performance of the BACKFORTH algorithm sensitive to the quality of the initialization. In this paper, a practical parthenogenetic algorithm PGMFL, which is a population-based meta-heuristic method, is proposed. The PGMFL algorithm can search multiple regions of a solution space simultaneously. A novel genetic operator is introduced based on the presented heuristic algorithm HF, which takes advantage of look-ahead mechanism and some potential information, i.e., the proportions of 0 and 1 entries in a column of recombinant matrix and those in the corresponding column of the founder matrix, and some other heuristic information, to compute the column values. The PGMFL algorithm can get fewer breakpoints and longer fragment average length than the BACKFORTH algorithm, which are proved by a number of experiments.**

***Index Terms*—founder, reconstruction, parthenogenetic algorithm**

I. Introduction

With the rapid development of sequencing technology, abundant DNA sequences and haplotyped SNP sequences are available. It has become possible to investigate the genealogy of a population, and genealogy inference, which is significant for discovering the genetic basis of complex diseases, has become one of the main topics in genomics [1][2]. It is widely believed that mutation can make the gene inherited from the ancestor to be a diseased one. It is expected that we can reconstruct the ancestral genomes from their offspring diseased ones, infer the pathogenesis according to their evolutionary processes and finally find the therapeutic methods.

Given a set of sample sequences from individuals of a population (for example, humans), the population descends from a small number of ancestral sequences called *founders*. Many findings from biological studies support the validity of this hypothesis. For example, the Ferroplasma type II genome seems to be a composite from three ancestral strains that have undergone homologous recombination to form a large population of mosaic genomes [3]. Researchers may try to construct more sample sequences (called *recombinants*) from relatively fewer founders so as to infer the evolutionary history of the recombinants. However, the number of founder sequences, as well as the founder sequences themselves, are generally unknown. The founder sequence reconstruction problem (FSRP), i.e., reconstructing a biologically plausible founder set or a recombinant mosaic pattern from a group of recombinant samples [4], has been formulated as a combinatorial optimization problem and received attention in computational biology today.

Since a large number of founder sets or possible mosaic patterns can be inferred from a set of sample sequences, biologically meaningful computational models are needed for inferring breakpoints and founder sequences. In 2002, Ukkonen [5] presented two computational models for the FSRP problem based on a mosaic model and a parsimony criterion: (1) the minimum founder set (MFS) model; (2) the maximum fragment length (MFL) model. The latter model is studied in this paper.

The maximum fragment length model is NP-hard in general case [6]. Recently it has been studied in some related works [5, 7-10]. Ukkonen [5] proposed a dynamic programming algorithm for solving this model. However, this algorithm does not scale well when the number of founders or the number/length of the recombinants grows. In 2007, Wu et al. [7] presented a tree search based algorithm RECBLOCK. The RECBLOCK algorithm lists all of the solutions by using exhaustive method, hence it has very good performance when the number of founders is small, but does not scale well either with the increase of founder number. In 2009, Roli et al. [8] developed a tabu search (TS) algorithm to solve the maximum fragment length model. The TS algorithm still works well for solving large size problems. In 2010, Benedettini et al. [9] presented a randomized iterated greedy algorithm

doi:10.4304/jcp.8.11.2934-2941

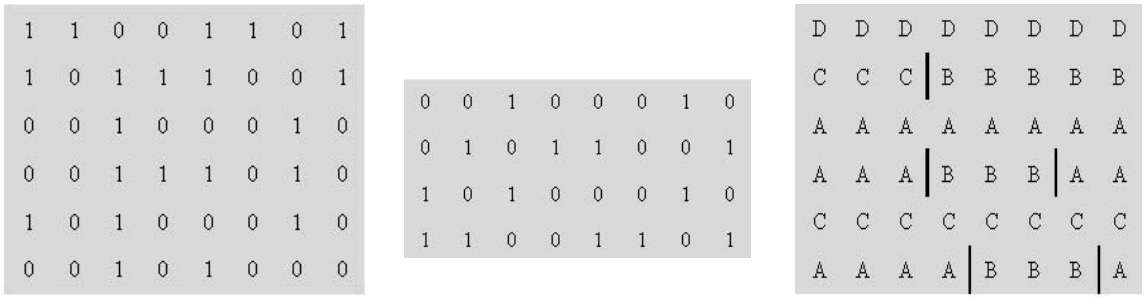
BACKFORTH, which has better performance than the TS algorithm proposed by Roli et al. [8]. In 2011, Blin et al. [10] put forward an accurate exponential algorithm. However, when the number of founder sequences increases, the performance of the algorithm still deteriorates.

The BACKFORTH algorithm adopts a meta-heuristic framework based on iterative greedy algorithm. It starts with a complete initial solution, and iteratively alternates between the partial destruction of the incumbent solution and the reconstruction of the resulting partial solution in order to obtain again a complete solution [9]. Iterative greedy algorithm is an optimization mechanism based on a single initial solution, hence it is very sensitive to the quality of the initialization. In this paper, a practical parthenogenetic algorithm PGMFL, which is a population-based meta-heuristic method, is proposed. The PGMFL algorithm can search multiple regions of a solution space simultaneously and it is relatively insensitive to the quality of the initialization. Experimental results show that within the same running time budget, the PGMFL algorithm can obtain better performance than the BACKFORTH algorithm under different parameter settings.

The rest of the paper is organized as follows. In next section, the maximum fragment length model is formalized, followed by a proposal of the PGMFL algorithm in section 3. The comparisons and analyses of the BACKFORTH algorithm and the PGMFL one are presented in section 4. Finally, some conclusions are drawn in section 5.

II. The Maximum Fragment Length Model

Given a set of m recombinants $C=\{C_1,C_2,\ldots,C_m\}$, where each recombinant C_i (i=1,2,…,m) is denoted by a string of length n over a given alphabet Σ, i.e., $C_i=c_{i1}c_{i2}\ldots c_{in}$ with $c_{ij}\in \Sigma$ (j=1,2,…,n). In this work, a typical biological application is concerned where the recombinants are haplotyped SNP sequences and $\Sigma=\{0,1\}$. The two symbols 0 and 1 encode the two most common alleles of each SNP site. As shown in Fig.1(a), each row denotes a recombinant and each column denotes a SNP site. Let $F=\{F_1,F_2,\ldots,F_k\}$ denote a set of k founders. Each founder $F_t=f_{t1}f_{t2}\ldots f_{tn}$ (t=1,2,…,k) is a gene sequence of length n with $f_{tj}\in \Sigma$ (j=1,2,…,n). As shown in Fig.1(b), each row denotes a founder while each column denotes a SNP site.



(a) (b) (c)

Figure 1. An example of the maximum fragment length model on binary sequences (a) illustrates a set of six recombinants, denoted by C1, C2, C3, C4, C5, and C6, respectively. (b) shows a set of four founders, denoted by A, B, C, and D, respectively. (c) shows a kind of mosaic structure for the recombinants in (a), and the vertical lines mark the breakpoints.

If a recombinant $C_i\in C$ (i=1,2,…,m) can be fragmented into p_i ($1\le p_i\le n$) non-empty strings (fragments) FR_{i1}, FR_{i2},…, FR_{ip_i}, such that each FR_{ij}(j=1,2,…, p_i) occurs in at least one sequence of F exactly at the same position as in C_i, C_i is regarded as reconstructing from F. Here, a breakpoint is recorded if two consecutive fragments FR_{ij} and FR_{ij+1} come from different founders. Take the fourth recombinant in Fig.1(a) for example, it can be decomposed into three fragments 001,110 and 10 by two breakpoints, and the three fragments respectively occur at the same position in the first, the second and the first founders (i.e. A and B resp.) as in the fourth recombinant (i.e. C4). In this case, we call the fourth recombinant can be reconstructed by the founder set in Fig.1(b).

If each recombinant $C_i\in C$ (i=1,2,…,m) can be reconstructed from founder set F, the set of recombinants C can be reconstructed from set F, and F is called a *founder set* of C. Given all of the fragments FR_{ij} (i=1,2,…,m, j=1,2,…, p_i), the number of breakpoint nb and the average fragment length λ_{ave} are defined as Formula (1) and Formula (2).

$$nb=\sum_{i=1}^{m} p_i - m, \tag{1}$$

$$\lambda_{ave}=\frac{m\times n}{nb}. \tag{2}$$

Based on the above concepts, a widely accepted optimization model was presented by Ukkonen [5] for the founder sequence reconstruction problem, as follows:

The maximum fragment length (MFL) model: given a set of recombinants C and a bound K for the number of founders, to find the largest λ_{ave} or the smallest nb that

are possible in a parse of C in terms of a founder set F of size at most K.

III. Algorithms for Solving the MFL Model

In this section, a constructive heuristic algorithm HF is presented for solving the maximum fragment length model. By using a novel genetic operator, which is based on the HF algorithm, a parthenogenetic algorithm PGMFL is proposed.

A. HF Algorithm

An $m\times n$ matrix is defined for recording m recombinant sequences of length n. For convenience of description, the recombinant matrix is still denoted by C, where each row denotes a recombinant, and each column denotes a SNP site. Similarly, a $K\times n$ founder matrix F is defined, where each row denotes a founder, and each column also denotes a SNP site.

The HF algorithm reconstructs the columns of founder sequences one after another. Following some notations used in the HF algorithm are defined. The column currently being constructed is called *current column*. For each row $C[i,\text{-}]$ (i=1,2,... ,m), we denote by $lb(i)$ the position of the last breakpoint in the row, and $df(i)$ the index of the founder that represents row i between the last breakpoint $lb(i)$ and *current column*. Take the fourth recombinant in Fig.1(a) for example, assume that *current column* is the fifth column, then $lb(4)$=3, and $df(4)$=2. For each row $F[t,\text{-}](t=1,2,\ldots,k)$, let $CS(t)$ be the set of all recombinants i with $df(i)=t$.

Let $S=\{s_{ij}\mid 1\le i\le d, 1\le j\le n\}$ be a set of d sequences of length n over alphabet Σ. We denote by $n_x(S,j)$ (x=0,1, j=1,2,... , n) the number of entries of column j that are equal to x (i.e. $\left|\{i\mid s_{ij}=x\}\right|$). Let $p_x(S,j)$ (x=0,1,j=1,2,... , n) be the proportion of x entries in the entries of columns j in matrix S, as shown in Formula (3).

$$p_x(S,j)=\frac{n_x(S,j)}{d} \tag{3}$$

Given matrices C and F, $ES_x(j)$ is defined as the ratio between $p_x(F,j)$ and $p_x(C,j)$, as defined in Formula (4).

$$ES_x(j)=\frac{p_x(F,j)}{p_x(C,j)},\quad x=0,1,\quad j=1,2\ldots,n \tag{4}$$

We, now, define a constructive algorithm HF. The input is an $m\times n$ recombinant matrix C, a founder matrix F, a bound K for the number of founders, a random control parameter rdp ($0\le rdp\le 1$), and the look-ahead parameters nt and las. The output is a founder matrix F. Some important steps of HF algorithm will be introduced in details afterwards.

1) Computing the first column: If the input founder matrix F is an empty one, i.e., there is not a value assigned to $F[i,j]$ (i=1,2,…,K, j=1,2,..., n), the HF algorithm starts by constructing the first column of matrix F as follows.

(a) Initialize counters $n_0(F,1)$ and $n_1(F,1)$ to 1 to ensure that at least one '0' entry, respectively one '1' entry, exists in the first column of the founder matrix F.

(b) Generate a random number r between 0 and 1, increase $n_0(F,1)$ by 1 when $r\le p_0(C,1)$, and increase $n_1(F,1)$ by 1 otherwise. The process is repeated for K-2 times.

(c) Initialize $F[1,1]$, $F[2,1]$, …, $F[n_0(F,1),1]$ to 0, and initialize $F[n_0(F,1)+1,1]$, $F[n_0(F,1)+2,1]$, …, $F[K,1]$ to 1.

2) Computing the remaining columns: Assume that the first j-1 columns of matrix F have already been constructed, and denote by F^{j-1} the corresponding partial solution. Update the variables $lb(i)$ and $df(i)$ (i=1,2,…,m), i.e., among all of the founders with $F[l,s]=C[i,s]$(l=1,…,K, s=h,…, j), select the founder lm with the minimum h as the new representant of recombinant $C[i,\text{-}]$, and set $lb(i)=h$, $df(i)=lm$.

Generate a random permutation $(r_1,r_2,\ldots,r_K)$ of the integers {1, 2, ..., K}, and the j-th column $F[\text{-},j]$ (j=2, 3, …, n) is constructed starting from row r_1 to r_K. Here we adopt look-ahead mechanism, which depends on the following two parameters:

(a) nt, the number of trials

(b) las, the look-ahead size

For this purpose, nt matrices $\{M_1,M_2,\ldots,M_{nt}\}$, each of which is composed of K rows and $\min\{las,\ n\text{-}j\text{+}1\}$ columns, are generated. Here each matrix M_d (d=1,…,nt) denotes a possible extension of F^{j-1} by $\min\{las,\ n\text{-}j\text{+}1\}$ columns. Compute b_d (d=1,…,nt), which is the optimal number of breakpoints obtained by appending M_d to the partial solution F^{j-1}. Set $F[\text{-},\ j]=M_D[\text{-},1]$, where D=argmin$\{b_d\}$. The concrete method for computing $M_d[r_k,\ l]$ (d=1,…,nt, k=1,…,K, l=1,…, $\min\{las,\ n\text{-}j\text{+}1\}$) is defined as follows.

Actually, M_d is computed by using a duplicate copy of matrix C. For convenience of description, this copy is still denoted by C. A random number r is drawn uniformly at random from [0,1]. If $r<rdp$, $M_d[r_k,\ l]$ is set to 0 with probability $p_0(CS(r_k),j\text{+}l\text{-}1)$, and to 1 otherwise. If $r\ge rdp$, $M_d[r_k,\ l]$ is computed with the following methods.

(a) If $n_0(CS(r_k),\ j\text{+}l\text{-}1)$ is greater (resp. less) than $n_1(CS(r_k),\ j\text{+}l\text{-}1)$, $M_d[r_k,l]$ is set to 0 (resp. 1).

(b) If $n_0(CS(r_k),\ j\text{+}l\text{-}1)$ has the same value as $n_1(CS(r_k),\ j\text{+}l\text{-}1)$, and $ES_0(j\text{+}l\text{-}1)$ is less (resp. greater) than $ES_1(j\text{+}l\text{-}1)$, $M_d[r_k,l]$ is set to 0 (resp. 1).

(c) If none of the above two cases is satisfied, $M_d[r_k,l]$ is set to 0/1 uniformly at random.

After $M_d[r_k,l]$ is assigned, the representant $df(i)$ of the recombinant $C[i,\text{-}]$ (i =1,2,…,m), which can not be represented anymore by its current representant (founder), need to be updated. That is to say, if $M_d[r_k,l]$=0 (resp. 1), the recombinant of C with $C[i,\ j\text{+}l\text{-}1]$=1(resp. 0) and $df(i)=r_k$ need to be reassigned representatives as follows: if there exists such a new representing founder r_{fl} (fl=k+1,…,K) that $C[i,s_1]=F[r_{fl},s_1]$ and $C[i,j\text{+}s_2\text{-}1]=M_d[r_{fl},s_2]$ (s_1=$lb(i)$,…,j-1, s_2=1,…,l-1), $df(i)$ is set to $r_{fl.}$, and the search process stops, otherwise $df(i)$ is unchanged.

After all the entries in the l-th (l=1,…, min{las, n-j+1}) column of matrix M_d are filled, some recombinants C[i,-] (i=1,2,…,m) can not be represented by their current founders, i.e., $C[i, j+l-1] \neq M_d[df(i), l]$. In such a situation, among all of the founders with $F[fl,s_1]=C[i,s_1]$ and $M_d[fl,s_2]= C[i,j+s_2-1]$ ($s_1=h,\ldots,j-1, s_2=1,\ldots,l$, fl=1,…,K), select the founder lm with the minimum h as the new representant of recombinant C[i,-], and set $lb(i)=h$, $df(i)=lm$.

Based on the above mentioned steps, HF algorithm is summarized in Fig. 2.

Algorithm 1: HF Algorithm

Input: C, F, K, rdp, nt, las
Output: F

```
1.  if F is an empty matrix then
2.      n0(F,1)=1, n1(F,1)=1
3.      for i=1,…,K-2 do
4.          generate a random number r (0≤r≤1)
5.          if (r≤p0(C,1)) then n0(F,1)++
6.          else n1(F,1) ++
7.      F[1,1]=,…,=F[n0(F,1),1]=0
8.      F[n0(F,1)+1,1] =,…,=F[K,1]=1
9.  update lb(i), df(i) (i=1,2,…,m)
10. for j=nc+1,…,n do // nc denotes the number of constructed columns in matrix F
11.     generate permutation (r1,r2,…,rK) (ri,i=1, 2,…,K) at random
12.     for d=1,...,nt do
13.         for l=1,…, min{las, n-j+1} do
14.             for k=1,...,K do
15.                 generate a random number r
16.                 if (r<rdp) then
17.                     Md[rk, l]=0 with probability p0(CS(rk),j+l-1), and Md[rk, l]=1 otherwise
18.                 else
19.                     if (n0(CS(rk), j+l-1)> n1(CS(rk), j+l-1)) then Md[rk,l]=0
20.                     else if (n0(CS(rk), j+l-1)< n1(CS(rk), j+l-1)) then Md[rk,l]=1
21.                     else if ES0(j+l-1)< ES1(j+l-1) then Md[rk,l]=0
22.                     else if ES0(j+l-1)> ES1(j+l-1) then Md[rk,l]=1
23.                     else Md[rk,l]=rand()%2 //rand() is used to generate a random integer
24.                 if (C[i, j+l-1]≠Md[rk,l] and df(i)=rk) then update df(i) (i=1,2,…,m)
25.             if (C[i, j+l-1]≠Md[df(i), l]) then update lb(i) and df(i) (i=1,2,…,m)
26.         Compute bd
27.     F[-,j]=MD[-,1] //D=argmin{bd}( d=1,...,nt)
```

Figure 2. HF Algorithm

B. PGMFL Algorithm

Parthenogenetic algorithm (PGA)[11][12] is a variant of genetic algorithm (GA) [13][14]. It employs gene recombination and selection operators instead of the traditional crossover operator to produce offspring. In the following, we will give some key techniques in designing PGMFL algorithm.

1) Construction of the hypothesis space: A binary matrix is used to denote a chromosome which represents a founder matrix of the given recombinant matrix. Thus, all of the binary matrices with K rows and n columns constitute the hypothesis space: $H=\{X_{K\times n} \mid X[i,j] \in \{0, 1\}, i=1,2,...,K, j=1,2,...,n\}$.

2) Generation of the initial population: A population is composed of N chromosomes, *i.e.* the population size is N. We generate an initial population by using HF algorithm with parameters nt=1 and las=1, i.e., look-ahead mechanism is not used while generating the initial population.

3) Selection operator: Both roulette wheel selection [15] and elitist model [16] are used to generate a new population for the next generation. The operator ensure that the best individual can be preserved into the next generation.

4) Recombination operator: Recombination operator generates new regions for search, that is generating new offspring chromosomes. Here a novel one based on HF algorithm is introduced, which has the following two characteristics.

(a) The recombination operator plants some random information into the chromosome. This can make the algorithm escape from local optimum solution and improve the clime ability of the algorithm.

(b) The recombination operator uses the information in the recombinants of matrix C to adjust the chromosome. This can improve the adaptability of the chromosome and the convergence speed of the algorithm simultaneously.

The idea of the recombination operator is based on the fact that solutions to a problem instance can be constructed from left to right, as explained in the HF algorithm, but also from right to left. Given a chromosome X, a binary variable *direction* is generated randomly. If *direction*=0, *dc* (a random integer between 1 and n) columns are removed from the right hand side or from the left hand side otherwise, i.e., the entries of these columns are deleted. Next, the HF algorithm is executed on the partial solution X to make it be a complete solution, and the new chromosome X' is obtained. The concrete description of the recombination operator is given in Fig. 3.

Algorithm 2: Recombination operator

Input: an $m \times n$ recombinant matrix C, a chromosome X
Output: a new chromosome X'
1. generate variable *direction* and *dc* randomly
2. **if** *direction*=0 **then**
3. **for** j=1 **to** *dc*
4. $X[i,j]$=' '(i=1,...,K) // $X[i,j]$ is deleted
5. **else if** *direction*=1 **then**
6. **for** j=1 **to** *dc*
7. $X[i,n-j+1]$=' '(i=1,...,K) // $X[i,n-j+1]$ is deleted
8. the HF algorithm is executed on X to generate X'

Figure 3. Recombination operator

4) Designation of the fitness function: The fitness function measures how "good" a chromosome is, so that better chromosomes can be selected. Because each chromosome represents a viable solution of the maximum fragment length model, we need to make an estimate of the results. Given a chromosome X and the recombinant matrix C, the fitness function Fitness(X) is defined as Formula (5):

$$Fitness(X) = \frac{1}{nb} \tag{5}$$

As mentioned above, chromosome X denotes a founder matrix of the given recombinant matrix, *nb* denotes the number of breakpoints in recombinant matrix C when given the founder matrix X. The smaller *nb* is, the stronger the fitness of chromosome X is.

The PGMFL algorithm for solving the founder sequences reconstruction problem is summarized in Fig.4.

IV. Experimental Results

In this section, experimental results are presented to compare the performance of the PGMFL and BACKFORTH algorithms. All experiments were performed on a Lenovo Workstation with Pentium(R) D 3.0GHz Processors and 2GB of RAM. The operating system was Windows XP Professional and the compiler was Microsoft Visual C++ 2010.

Algorithm 3: PGMFL Algorithm

Input: C, F, K, *rdp*, *nt*, *las*, N, *maxtime* //*maxtime* is the maximum of running time
Output: F
1. Generate the initial population $pop_0=\{X_1(0),\ldots,X_M(0)\}$
2. **for** i=1,…,N **do**
3. compute Fitness($X_i(0)$)
4. tm=0 //tm denotes the actual iteration time
5. gen=0 // gen denotes the actual iteration times
6. **If** tm<*maxtime* **then**
7. select N members from pop_{gen} by using selection operator, and add them to pop_{gen+1}
8. **for** i=1,…,N **do**
9. apply recombination operator on $X_i(gen+1)$ to produce $X_i'(gen+1)$
10. **If** Fitness($X_i'(gen+1)$)> Fitness($X_i(gen+1)$) **then**
11. $X_i(gen+1)= X_i'(gen+1)$
12. get the value of tm
13. $gen=gen+1$
14. output the individual with the highest fitness

Figure 4. PGMFL Algorithm

In the experiments, we used the following two measurements to evaluate the performance of the two algorithms:

(a) *nb*, the number of breakpoints
(b) λ_{ave}, the average fragment length

A similar benchmark set introduced in Ref. [8] and Ref. [9] was used. This set is composed of randomly generated instances with $m \in \{30, 50, 100\}$ recombinants and $n \in \{2m, 3m, 5m\}$ SNP sites. The best set of parameters given in Ref. [9] was used for the BACKFORTH algorithm. The parameters of the PGMFL algorithm were set as follows: *rdp*=0.2, *nt*=10 (5 in case k=3), *las*=1, N=300, which are the same as the BACKFORTH algorithm. For every experimental instance, we have limited the CPU time to one hour, which is the same as Ref. [9].

Table I to Table III show the experimental results of instances with 30 recombinant sequences, where the number of SNP sites n is set to 60, 90 and 150

respectively. In each of the three tables, eight sets of parameters were set in dealing with the bound K for the number of founders.

From Table I to Table III we can see that with the increase of parameter K, the nbs obtained by the two algorithms are both decreased. In Table I, the nb of algorithm BACKFORTH ranges from 611 to 260, and the one of algorithm PGMFL ranges from 595 to 247. In Table II, the nb of algorithm BACKFORTH ranges from 923 to 402, and the one of algorithm PGMFL ranges from 914 to 399. In Table III, the nb of algorithm BACKFORTH ranges from 1590 to 754, and the one of algorithm PGMFL ranges from 1555 to 747.

TABLE I.
PERFORMANCE COMPARISONS WITH M=30, N=60.

K	λ_{ave}		Nb	
	BACKFORTH	PGMFL	BACKFORTH	PGMFL
3	2.95	3.03	611	595
4	3.57	3.62	504	497
5	4.15	4.17	434	432
6	4.64	4.84	388	372
7	5.28	5.64	341	319
8	5.83	6.12	309	294
9	6.43	6.50	280	277
10	6.92	7.29	260	247

TABLE II.
PERFORMANCE COMPARISONS WITH M=30, N=90.

K	λ_{ave}		Nb	
	BACKFORTH	PGMFL	BACKFORTH	PGMFL
3	2.93	2.95	923	914
4	3.52	3.62	768	746
5	4.07	4.28	664	631
6	4.70	4.75	575	569
7	5.14	5.23	525	516
8	5.57	5.78	485	467
9	6.04	6.18	447	437
10	6.72	6.77	402	399

TABLE III.
PERFORMANCE COMPARISONS WITH M=30, N=150.

K	λ_{ave}		Nb	
	BACKFORTH	PGMFL	BACKFORTH	PGMFL
3	2.83	2.89	1590	1555
4	3.49	3.52	1290	1278
5	4.11	4.13	1095	1089
6	4.55	4.62	988	975
7	5.07	5.09	887	884
8	5.48	5.53	821	814
9	5.97	6.02	754	747
10	6.39	6.47	704	696

It can be known from Formula (2) that the value of λ_{ave} is inversely proportional to the value of nb. The bigger nb is, the smaller λ_{ave} is. Therefore, with the increase of parameter K, the average fragment length λ_{ave} obtained by the two algorithms is increased. In Table I, the λ_{ave} of algorithm BACKFORTH ranges from 2.95 to 6.92, and the one of algorithm PGMFL ranges from 3.03 to 7.29. In Table II, the λ_{ave} of algorithm BACKFORTH ranges from 2.93 to 6.72, and the one of algorithm PGMFL ranges from 2.95 to 6.77. In Table III, the λ_{ave} of algorithm BACKFORTH ranges from 2.83 to 6.39, and the one of algorithm PGMFL ranges from 2.89 to 6.47.

In addition, in these three tables, with the same running time budget, the PGMFL algorithm can get fewer breakpoints and longer average fragment length than the BACKFORTH algorithm under different parameter settings.

Table IV to Table VI show the experimental results of instances with 50 recombinants, where n is set to 100, 150 and 250 respectively. In each of the three tables, eight sets of parameters were also set in dealing with parameter K.

TABLE IV.
PERFORMANCE COMPARISONS WITH M=50, N=100.

K	λ_{ave}		Nb	
	BACKFORTH	PGMFL	BACKFORTH	PGMFL
3	2.70	2.74	1854	1828
4	3.26	3.27	1536	1530
5	3.60	3.71	1387	1348
6	4.07	4.15	1228	1206
7	4.48	4.51	1116	1108
8	4.84	4.93	1033	1014
9	5.27	5.29	949	946
10	5.61	5.64	891	887

TABLE V.
PERFORMANCE COMPARISONS WITH M=50, N=150.

K	λ_{ave}		Nb	
	BACKFORTH	PGMFL	BACKFORTH	PGMFL
3	2.71	2.73	2765	2746
4	3.16	3.20	2371	2346
5	3.54	3.63	2116	2068
6	4.03	4.04	1861	1857
7	4.38	4.47	1711	1678
8	4.81	4.84	1560	1548
9	5.13	5.15	1461	1457
10	5.44	5.47	1379	1370

TABLE VI.
PERFORMANCE COMPARISONS WITH M=50, N=250.

K	λ_{ave}		Nb	
	BACKFORTH	PGMFL	BACKFORTH	PGMFL
3	2.65	2.67	4710	4678
4	3.12	3.14	4001	3985
5	3.58	3.59	3489	3478
6	3.97	3.99	3145	3132
7	4.31	4.34	2902	2883
8	4.70	4.73	2657	2644
9	5.04	5.05	2481	2477
10	5.36	5.37	2330	2326

Table IV to Table VI show that with the increase of parameter K, the number of breakpoints nb obtained by the two algorithms is decreased. In Table IV, the nb of algorithm BACKFORTH ranges from 1854 to 891, and the one of algorithm PGMFL ranges from 1828 to 887. In Table V, the nb of algorithm BACKFORTH ranges from 2765 to 1379, and the one of algorithm PGMFL ranges from 2746 to 1370. In Table VI, the nb of algorithm BACKFORTH ranges from 4710 to 2330, and the one of algorithm PGMFL ranges from 4678 to 2326.

As mentioned above, in these three tables, with the increase of parameter K, the average fragment length λ_{ave} obtained by the two algorithms is also increased. In Table

IV, the λ_{ave} of algorithm BACKFORTH ranges from 2.70 to 5.61, and the one of algorithm PGMFL ranges from 2.74 to 5.64. In Table V, the λ_{ave} of algorithm BACKFORTH ranges from 2.71 to 5.44, and the one of algorithm PGMFL ranges from 2.73 to 5.47. In Table VI, the λ_{ave} of algorithm BACKFORTH ranges from 2.65 to 5.36, and the one of algorithm PGMFL ranges from 2.67 to 5.37.

In Table IV to Table VI, the PGMFL algorithm still has better performance than the BACKFORTH algorithm under different parameter settings with the same running time budget.

In Table VII, four sets of parameters were set in dealing with parameter K: (1) K=3, (2) K=4, (3) K=5, (4) K=6. In all of these four cases, the number of recombinants m was set to 100, and the number of SNP sites n was set to 200, 300 and 500 respectively.

TABLE VII.
PERFORMANCE COMPARISONS WITH M=100.

K	*n*	λ_{ave}		*Nb*	
		BACKFORTH	**PGMFL**	**BACKFORTH**	**PGMFL**
3	200	2.54	2.55	7860	7843
	300	2.52	2.53	11927	11874
	500	2.51	2.51	19939	19894
4	200	2.89	2.95	6921	6780
	300	2.88	2.92	10416	10285
	500	2.86	2.91	17459	17186
5	200	3.23	3.27	6187	6109
	300	3.23	3.28	9297	9135
	500	3.20	3.27	15613	15298
6	200	3.59	3.60	5566	5551
	300	3.58	3.64	8386	8231
	500	3.52	3.59	14216	13938

From Table VII we can see that within the same running time budget, the PGMFL algorithm can obtain fewer breakpoints and longer average fragment length than the BACKFORTH algorithm under all parameter settings. In addition, given a certain value of bound K, with the increase of SNP sites n, although the absolute number of breakpoints obtained by the PGMFL algorithm and the BACKFORTH algorithm increases, the average fragment length got by these two algorithms keeps almost unchanged. Hence, the parameter n has little effect on the performance of the PGMFL and the BACKFORTH algorithms.

The above experimental results show that the PGMFL algorithm is able to get better performance than the BACKFORTH algorithm. During the iteration process, the BACKFORTH algorithm adopts a meta-heuristic framework based on iterative greedy algorithm and makes full use of the effective information of the obtained solutions. The BACKFORTH algorithm starts with a single initial solution, and iteratively refines it to get the optimal solution. However, the BACKFORTH algorithm uses an optimization mechanism based on a single initial solution, hence the performance of the BACKFORTH is sensitive to the quality of the initial solution. The PGMFL algorithm is based on parthenogenetic algorithm, which can search multiple regions of a solution space simultaneously and it is relatively insensitive to the quality of the initial solution. In addition, the recombination operator based on the HF algorithm makes full use of the information in the recombinant sequences to adjust the chromosomes step by step, which plays a strong role in evolving towards higher adaptability for chromosomes. The operator still plants random information for evolution, which prevents premature convergence to local optima and improves the clime ability of the PGMFL algorithm. Therefore, the PGMFL algorithm can obtain a better solution than the BACKFORTH algorithm.

V. CONCLUSIONS

In recent years, the problem of reconstructing founder sequences from a set of recombinant sequences has received much attention in computational biology. The maximum fragment length model is one of the important computational models for solving the founder sequences reconstruction problem.

In this paper, a parthenogenetic algorithm PGMFL is presented for solving the model by introducing an effective recombination operator based on the HF algorithm. Comparing with the BACKFORTH algorithm, the PGMFL algorithm can get better solutions within the same time budget, which were tested by a number of experiments. Furthermore, the PGMFL algorithm has high efficiency and is intended to reconstruct long founders. In conclusion, the PGMFL algorithm is a practical solution for solving the founder sequences reconstruction problem.

ACKNOWLEDGMENT

The authors are grateful to Professor Yufeng Wu for his kindly providing the ADH sample data, and Professor Qi Zhang for bringing the minimum mosaic problem to our attention. This research was supported in part by Guangxi Natural Science Foundation under Grant No. 2011GXNSFB018068, No. 2011GXNSFB018070 and No. 2012GXNSFAA053219, the National Natural Science Foundation of China under Grant No. 61165009, and "Bagui Scholar" Project Special Funds.

REFERENCES

[1] J. D. Kececioglu, D. Gusfield, "Reconstructing a history of recombinations from a set of sequences", Discrete Applied Mathematics, vol.88, no.1-3, pp.239-260,1998.

[2] T. Pupko, I. Pe'er, R. Shamir, D. Graur, "A fast algorithm for joint reconstruction of ancestral amino acid sequences", Molecular Biology and Evolution, vol.17, pp.890-896,2000.

[3] G. Tyson, J. Chapman, P. Hugenholtz, et al., "Community structure and metabolism through reconstruction of microbial genomes from the environment", Nature, vol.428, pp.37-43,2004.

[4] Q. Zhang, W. Wang, L. McMillan, F.P.D. Villena, D.Threadgill, "Inferring genome-wide mosaic structure", In Proceeding of the Pacific Symposium on Biocomputing, pp.150-161,2009.

[5] E. Ukkonen, "Finding founder sequences from a set of recombinants", Lecture Notes in Computer Science, vol.2452, pp.277-286,2002.
[6] P. Rastas, E. Ukkonen, "Haplotype inference via hierarchical genotype parsing", Lecture Notes in Computer Science, vol.4645, pp.85-97,2007.
[7] Y. Wu, D. Gusfield, "Improved algorithms for inferring the minimum mosaic of a set of recombinants", Lecture Notes in Computer Science, vol.4580, pp.150-161,2007.
[8] A. Roli, C. Blum, "Tabu search for the founder sequence reconstruction problem: a preliminary study", Lecture Notes in Computer Science, vol.5518, pp.1035-1042,2009.
[9] S. Benedettini, C. Blum, A. Roli, "A randomized iterated greedy algorithm for the founder sequence reconstruction problem", Lecture Notes in Computer Science, vol.6073, pp.37-51,2010.
[10] G. Blin, R. Rizzi, F. Sikora, S. Vialette, "Minimum mosaic inference of a set of recombinants", In Proceedings of the 17th Computing: the Australasian Theory Symposium, pp.23-30,2011.
[11] M. J. Li, T. S. Tong, "A partheno-genetic algorithm and analysis on its global convergence", Acta Automatic Sinica, vol.25, pp.68-72,1999.
[12] J. L. Wu, J. X. Wang, J. E. Chen, "A parthenogenetic algorithm for single individual SNP haplotyping", Engineering Applications of Artificial Intelligence, vol.22, pp.401-406,2009.
[13] M. Mitchell, "Genetic algorithms: an overview", Complexity, vol.1, no.1, pp.31-39,1995.
[14] S. A. Ghoreishi, M. A. Nekoui, S. Partovi, S. O. Basiri, "Application of Genetic Algorithm for Solving Multi-Objective Optimization Problems in Ro-bust Control of Distillation Column", International Journal of Advancements in Computing Technology, vol.3, no.1, pp.32-43, 2011.
[15] R.Sivaraj, Dr.T.Ravichandran, "A review of selection methods in genetic algorithm", International Journal of Engineering Science and Technology, vol.3, no.5, pp.3792-3797,2011.
[16] D.Bhandari, C.A.Murthy, "Genetic algorithm with elitist model and its convergence", International Journal of Pattern Recognition and Artificial Intelligence,vol.10,no.6,pp.731-747, 1996.

Jingli Wu, born in 1978, an associate professor of the College of Computer Science and Information Technology, Guangxi Normal University, Guilin, P.R. China. She received her Ph.D. degree in Computer Science from Central South University of P.R. China in 2008. Her current research interests include biocomputing, algorithm analysis and optimization.

Hua Wang, born in 1985, she is currently working towards his M.S. degree in College of Computer Science and Information Technology, Guangxi Normal University. Her research interests are in the areas of bio-computing and evolutionary algorithm