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 exhaustiv 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

© 2013 ACADEMY PUBLISHER
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 \( C_i \) \((i=1,2,\ldots,m)\) is denoted by a string of length \( n \) over a given alphabet \( \Sigma \), i.e., \( C_i = c_{i1}c_{i2}\ldots c_{in} \) with \( c_{ij} \in \Sigma \) \((j=1,2,\ldots,n)\). 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 \( F = f_{1j}f_{2j}\ldots f_{nj} \) \((i=1,2,\ldots,k)\) is a gene sequence of length \( n \) with \( f_{ij} \in \Sigma \) \((j=1,2,\ldots,n)\). As shown in Fig.1(b), each row denotes a founder while each column denotes a SNP site.

If a recombinant \( C_i \in C \) \((i=1,2,\ldots,m)\) can be fragmented into \( p_i \) \((1 \leq p_i \leq n)\) non-empty strings (fragments) \( FR_{ij}, FR_{i2}, \ldots, FR_{in} \), such that each \( FR_{ij} \) \((j=1,2,\ldots,p_i)\) occurs 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 100 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 \( C_i \in C \) \((i=1,2,\ldots,m)\) 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 \( FR_{ij} \) \((i=1,2,\ldots,m, j=1,2,\ldots,p_i)\), the number of breakpoint \( nb \) and the average fragment length \( \lambda_{av} \) are defined as Formula (1) and Formula (2).

\[
\begin{align*}
    nb &= \sum_{i=1}^{m} p_i - m, \\
    \lambda_{av} &= \frac{m \times n}{nb}
\end{align*}
\]

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 \( \lambda_{av} \) 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 current column being constructed is called current column. For each row $C[i,-]$ ($i=1,2,...,m$), 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 $F[l,-]$ ($l=1,2,...,K$), let $CS(l)$ be the set of all recombinants $j$ with $df(i)=l$.

Let $S=[s_{ij}] 1 \leq i \leq d, 1 \leq j \leq n$ be a set of $d$ sequences over alphabet $\Sigma$. We denote by $n_i(S,j)$ ($x=0,1$, $j=1,2,...,n$) the number of entries of column $j$ that are equal to $x$ (i.e. $|\{i|s_{ij}=x\}|$). Let $p_i(S,j)$ ($x=0,1$, $j=1,2,...,n$) be the proportion of $x$ entries in the entries of column $j$ in matrix $S$, as shown in Formula (3).

$$p_i(S,j)=\frac{n_i(S,j)}{d} \quad (3)$$

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

$$ES_i(j)=\frac{p_i(F,j)}{p_i(C,j)} \quad x=0,1, j=1,2,...,n \quad (4)$$

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 $rdp$ ($0 \leq rdp \leq 1$), 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 $r \leq p_0(C,1)$, and increase $n_1(F,1)$ by 1 otherwise. The process is repeated for $K-2$ times.
(c) Initialize $F[1,1], F[2,1],...,F[n_F,1]$ to 0, and initialize $F[n_F+1,1], F[n_F+2,1],...,F[K,1]$ 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 $F[i,s]=C[i,s]$ ($l=1,...,K$, $s=h,...,j$), select the founder $lm$ with the minimum $h$ as the new representant of recombinant $C[i,-]$, and set $lb(i)=h, df(i)=lm$.

Generate a random permutation $(r_1,r_2,...,r_K)$ of the integers $\{1, 2,...,K\}$, and the $j$-th column $F[-,j]$ ($j=2, 3,...,n$) 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 $\{M_1,M_2,...,M_{nt}\}$, each of which is composed of $K$ rows and min{$las, n-j+1$} columns, are generated. Here each matrix $M_j$ ($j=1,...,nt$) denotes a possible extension of $F^{j-1}$ by min{$las, n-j+1$} columns. Compute $b_j$ ($d=1,...,nt$), which is the optimal number of breakpoints obtained by appending $M_j$ to the partial solution $F^{j-1}$. Set $F[-,j]=M_0[-,1]$, where $D=\arg\min\{b_j\}$. The concrete method for computing $M_d[-,l]$ ($d=1,...,nt$, $k=1,...,K$, $l=1,...,\min\{las, n-j+1\}$) 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 $r=rdp$, $M_d[-,l]$ is set to 0 with probability $p_0(C[r,l])$, and 1 otherwise. If $r \neq rdp$, $M_d[-,l]$ is computed with the following methods.

(a) If $n_0(C[r,l], j+l-1)$ is greater (resp. less) than $n_i(C[r,l], j+l-1)$, $M_d[-,l]$ is set to 0 (resp. 1).
(b) If $n_0(C[r,l], j+l-1)$ has the same value as $n_i(C[r,l], j+l-1)$, and $ES_i(j+l-1)$ is less (resp. greater) than $ES_j(j+l-1)$, $M_d[-,l]$ is set to 0 (resp. 1).
(c) If none of the above two cases is satisfied, $M_d[-,l]$ is set to 0/1 uniformly at random.

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

After all the entries in the \(l\)-th \((l=1,\ldots, \min\{las, n+j+1\})\) column of matrix \(M_{fl}\) are filled, some recombinants \(C[i,\cdot]\) \((i=1,2,\ldots,m)\) can not be represented by their current founders, i.e., \(C[i, j+l+1] \neq M_{fl}[df(i), l]\). In such a situation, among all of the founders with \(F[fls_1]=C[i, s_1]\) and \(M_{fl}[fls_2]= C[i, j+s_2+1] \) \((s_1=h,\ldots, j+s_2+1=1,\ldots, l, fl=1,\ldots,K)\), select the founder \(lm\) with the minimum \(h\) as the new representant of recombinant \(C[i,\cdot]\), and set \(lb(i)=h, df(i)=lm\).

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

### Figure 2. HF Algorithm

<table>
<thead>
<tr>
<th>Algorithm 1: HF Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> C, F, K, rdp, nt, las</td>
</tr>
<tr>
<td><strong>Output:</strong> F</td>
</tr>
<tr>
<td>1. if (F) is an empty matrix then</td>
</tr>
<tr>
<td>2. (n_{F, 1}=1, n_{F, 1}=1)</td>
</tr>
<tr>
<td>3. for (i=1,\ldots, K, 2) do</td>
</tr>
<tr>
<td>generate a random number (r) ((0 \leq r \leq 1))</td>
</tr>
<tr>
<td>4. if ((r \leq p_0(C, 1))) then (n_{F, 1}++)</td>
</tr>
<tr>
<td>5. else (n_{F, 1}++)</td>
</tr>
<tr>
<td>6. (F[1, 1]=\ldots=F[n_{F, 1}], 1]=0)</td>
</tr>
<tr>
<td>7. (F[n_{F, 1}+1, 1]=\ldots=F[K, 1]=1)</td>
</tr>
<tr>
<td>8. update (lb(i), df(i)) ((i=1, 2,\ldots, m))</td>
</tr>
<tr>
<td>9. for (j=n_{F, 1}+1,\ldots, nt) do if (n_{j}) denotes the number of constructed columns in matrix (F)</td>
</tr>
<tr>
<td>generate permutation ((r_1, r_2,\ldots, r_K)) ((r_i=1, 2,\ldots, K)) at random</td>
</tr>
<tr>
<td>for (d=1,\ldots, nt) do</td>
</tr>
<tr>
<td>for (k=1,\ldots, K) do</td>
</tr>
<tr>
<td>generate a random number (r)</td>
</tr>
<tr>
<td>if ((r&lt;rdp)) then</td>
</tr>
<tr>
<td>(M_{fl}[r_1, l]=0) with probability (p_0(CS(r_1), j+l-1)), and (M_{fl}[r_1, l]=1) otherwise</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>if ((n_{F, 1}(CS(r_1), j+l)&gt; n_{F, 1}(CS(r_1), j+l-1))) then (M_{fl}[r_1, l]=0)</td>
</tr>
<tr>
<td>else if ((n_{F, 1}(CS(r_1), j+l-1)&lt; n_{F, 1}(CS(r_1), j+l))) then (M_{fl}[r_1, l]=1)</td>
</tr>
<tr>
<td>else if (ES_0(j+l+1)&lt; ES_0(j+l-1)) then (M_{fl}[r_1, l]=0)</td>
</tr>
<tr>
<td>else if (ES_0(j+l-1)&lt; ES_0(j+l-1)) then (M_{fl}[r_1, l]=1)</td>
</tr>
<tr>
<td>else (M_{fl}[r_1, l]=\text{rand}(0&lt;2)); (\text{rand}(\cdot)) is used to generate a random integer</td>
</tr>
<tr>
<td>if ((C[i, j+l+1] \neq M_{fl}[r_1, l]) and (df(i)=r_1) then update (df(i)) ((i=1, 2,\ldots, m))</td>
</tr>
<tr>
<td>else if ((C[i, j+l-1] \neq M_{fl}[df(i), l])) then update (lb(i)) and (df(i)) ((i=1, 2,\ldots, m))</td>
</tr>
<tr>
<td>Compute (h_2)</td>
</tr>
<tr>
<td>(F[\cdot, j]=M_{fs}[-1]) (/D=\text{argmin}{h_2}) ((d=1,\ldots, nt))</td>
</tr>
</tbody>
</table>

#### 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: \(H^c=\{X_{e_n} | X_{e_n} \in \{0, 1\}, i=1, 2,\ldots, K, j=1, 2,\ldots, n\}\).

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**

**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 $pop_0 = \{X_1(0),...,X_N(0)\}$
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) $\lambda_{ave}$, 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 $me \{30, 50, 100\}$ 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 $nb$s 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 821 to 399. In Table III, the $nb$ of algorithm BACKFORTH ranges from 1590 to 754, and the one of algorithm PGMFL ranges from 1278 to 664.

It can be known from Formula (2) that the value of $\lambda_{ave}$ is inversely proportional to the value of $nb$. The bigger $nb$ is, the smaller $\lambda_{ave}$ is. Therefore, with the increase of parameter $K$, the average fragment length $\lambda_{ave}$ obtained by the two algorithms is increased. In Table I, the $\lambda_{ave}$ of algorithm BACKFORTH ranges from 2.95 to 6.92, and the one of algorithm PGMFL ranges from 2.83 to 3.03. In Table II, the $\lambda_{ave}$ of algorithm BACKFORTH ranges from 4.07 to 5.14, and the one of algorithm PGMFL ranges from 3.57 to 3.62. In Table III, the $\lambda_{ave}$ of algorithm BACKFORTH ranges from 2.93 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 that with the increase of parameter $K$, the number of breakpoints $nb$ obtained by the two algorithms is also decreased. In Table IV, the $nb$ of algorithm BACKFORTH ranges from 1854 to 891, and the one of algorithm PGMFL ranges from 1461 to 1379.

As mentioned above, in these three tables, with the increase of parameter $K$, the average fragment length $\lambda_{ave}$ obtained by the two algorithms is also increased. In Table II, the $\lambda_{ave}$ of algorithm BACKFORTH ranges from 3.26 to 4.73, and the one of algorithm PGMFL ranges from 3.14 to 3.99.

Table IV to Table VI show that with the increase of parameter $K$, the number of breakpoints $nb$ obtained by the two algorithms is also decreased. In Table IV, the $nb$ of algorithm BACKFORTH ranges from 1854 to 891, and the one of algorithm PGMFL ranges from 1461 to 1379. In Table V, the $nb$ of algorithm BACKFORTH ranges from 2765 to 1379, and the one of algorithm PGMFL ranges from 2477 to 1206. In Table VI, the $nb$ of algorithm BACKFORTH ranges from 4710 to 2326, and the one of algorithm PGMFL ranges from 4678 to 2326.
IV, the $\lambda_{av}$ 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 $\lambda_{av}$ of algorithm BACKFORTH ranges from 2.71 to 5.44, and the one of algorithm PGMFL ranges from 2.73 to 5.37. In Table VI, the $\lambda_{av}$ 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>
<thead>
<tr>
<th>$K$</th>
<th>$n$</th>
<th>$\lambda_{av}$</th>
<th>$\mu_{av}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>200</td>
<td>2.54</td>
<td>2.55</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>2.52</td>
<td>2.53</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>2.51</td>
<td>2.51</td>
</tr>
<tr>
<td>4</td>
<td>200</td>
<td>2.89</td>
<td>2.95</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>2.88</td>
<td>2.92</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>2.86</td>
<td>2.91</td>
</tr>
<tr>
<td>5</td>
<td>200</td>
<td>3.23</td>
<td>3.27</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>3.23</td>
<td>3.28</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>3.20</td>
<td>3.27</td>
</tr>
<tr>
<td>6</td>
<td>200</td>
<td>3.59</td>
<td>3.60</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>3.58</td>
<td>3.64</td>
</tr>
<tr>
<td></td>
<td>500</td>
<td>3.52</td>
<td>3.59</td>
</tr>
</tbody>
</table>

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 recombiant 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. 2011GXNSFBO18068, No. 2011GXNSFB018070 and No. 2012GXNSFAA053219, the National Natural Science Foundation of China under Grant No. 61165009, and “Bagui Scholar” Project Special Funds.

REFERENCES


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.