中央研究院
資訊科學研究所
Institute of Information Science, Academia Sinica • Taipei, Taiwan, ROC

TR-IIS-09-011

GAPM – A Robust Algorithm for the Physical Mapping Problem

Hsin-Nan Lin, Yen-Yi Lin, Shinn-Ying Ho, Ting-Yi Sung, and Wen-Lian Hsu

December 29 2009 || Technical Report No. TR-IIS-09-011
http://www.iis.sinica.edu.tw/page/library/LIB/TechReport/tr2009/tr09.html

GAPM – A Robust Algorithm for the Physical Mapping Problem

Hsin-Nan Lin[1,2,3], Yen-Yi Lin[2], Shinn-Ying Ho[3], Ting-Yi Sung[2], and Wen-Lian Hsu[2,⋆]

[1] Taiwan International Graduate Program, Academia Sinica, Taipei, Taiwan
[2] Bioinformatics Lab., Institute of Information Science, Academia Sinica, Taipei, Taiwan
[3] Institute of Bioinformatics, National Chiao Tung University, Hsinchu, Taiwan
hsu@iis.sinica.edu.tw

Abstract. A major challenge for next generation sequencing technology is genome assembly. A physical map could be used as a preliminary step towards genome sequencing in a hybrid approach. In this paper, we illustrate a robust physical mapping algorithm, GAPM, which could well complement with the assembly of short fragments. The physical mapping problem (PMP) is to determine the relative positions of genetic markers (called probes) along the DNA sequences. The presence and absence of probes in clones can be represented by a 0-1 matrix with rows corresponding to clones and columns corresponding to probes. A 0-1 matrix satisfies the *consecutive ones property* (*COP*) for the rows if there exists a column permutation such that the ones in each row of the resulting matrix are consecutive. In the error-free case, the PMP can be reduced to testing the COP of a 0-1 matrix. Lu and Hsu proposed an iterative clustering algorithm to deal with the following four types of errors: false positives, false negatives, chimerical clones, and non-unique probes. In this paper, we present a novel genetic algorithm, called GAPM, with a much better performance. GAPM can be run in parallel and generate approximate optimal physical maps regardless of the error rates and matrix sizes. Moreover, GAPM is very flexible in dealing with unknown data. We test 9,000 different cases and compare GAPM with L&H's method. The results indicate that GAPM is more robust and reliable for most data.

Key words: sequence assembly, consecutive ones property, physical mapping, probe hybridization, genetic algorithms

1 Introduction

The fascinating next generation sequencing technologies provide ultra high throughput DNA sequencing. They produce a huge amount of sequence fragments in the range of 20-300 base pairs. However, as the number of DNA sequences is rapidly increasing, the new technologies present major bioinformatics challenges, particularly for genome assembly [1]. Many applications of next generation sequencing require anchoring of these fragments onto a reference sequence [2]. In the Plant and Animal Genome Meeting (PAG 2008), it was agreed that a robust physical map should be completed as a preliminary step towards genome sequencing using a hybrid approach: an optimized BAC pool sequencing strategy coupled with next generation sequencing technologies. In [3], the authors described a genus-wide comparative framework which is composed of BAC fingerprints and end-sequenced physical maps. This framework is highly compatible with next generation sequencing technologies whereby whole genomes can be sequenced in 4-8 Mb chunks [4]. In this paper, we illustrate a robust physical mapping algorithm, GAPM, which could well complement with the assembly of short fragments.

In DNA sequence analysis, the physical mapping problem is to determine the relative positions of genetic markers (called probes) along the DNA sequences. A probe is usually a unique sequence of a few hundred nucleotides. The resulting maps are used as the basis for DNA sequencing, and for the isolation and characterization of individual genes. The construction of a physical map is generally accomplished as follows. First, long DNA sequences are separated into smaller fragments (called clones). A number of probes are tested for their presence or absence in the clones. Given the collection of probes, one tries to order the probes in

⋆ To whom correspondence should be addressed.

such a way that probes attached to the same clone are consecutive. The presence and absence of probes in clones can be represented by a 0-1 matrix with rows corresponding to clones and columns corresponding to probes, where a 1 means presence and a 0 absence. A 0-1 matrix satisfies the consecutive ones property (COP) for the rows if we could find a column permutation such that the ones in each row are consecutive. The resulting column permutation reflects the ordering of probes that implies the relative positions of the clones for reconstructing the DNA sequences.

In the error-free case, the physical mapping problem can be transformed into the problem of finding a column permutation which satisfies the COP. Booth and Lueker proposed a linear time algorithm to determine whether or not a 0-1 matrix has the COP [5]. Subsequently, Hsu proposed another simpler linear time algorithm [6]. However, such discrete algorithms can hardly be adapted to data with errors. In the wet-lab experiments for probes hybridizing with clones, four types of experiment errors inevitably occur, namely, false positives, false negatives, non-unique probes and chimeric clones. A false positive is a probe falsely present in a clone. It will result in an entry of 1 that should be 0 in the 0-1 matrix. On the contrary, a false negative is a probe falsely absent from a clone that results in an entry of 0 that should be 1. A non-unique probe is a probe whose DNA sequence occurs more than once within a chromosome. It would bind to multiple clones which are far from each other. This error results in false overlaps among those clones. Two (or more) clones which connect at their ends form a chimeric clone. It results in a false row being the union of several rows in the 0-1 matrix.

In order to reconstruct a correct DNA sequence, errors need to be detected. Karp considered that the reassembly process of a physical map on a large DNA molecule leads to a number of challenging problems in computer science [7]. Several related problems have been proved to be NP-complete or NP-hard [8,9,10]. Cuticchia et al. provided a methodology for quickly ordering random clones into a physical map using the amount of overlaps between any two clones [11]. Alizadeh et al. suggested using maximum-likelihood functions to model the physical mapping problem and solved this problem based on the local search [12]. Jain and Myers converted the physical mapping problem into a 0/1 linear programming problem [13]. Mayraz and Shamir constructed physical maps using a greedy method based on a Bayesian overlap score [14]. Goncalves et al. applied the Bayes' theorem to evaluate corresponding posterior probabilities on this problem for reconstructing the circular genome sequences [15]. Ukkonen narrowed down this problem to find the partial orders from a unordered 0-1 matrix [16]. Other methods mainly focused on removing typical errors such as false positives and chimeric clones [17]. However, none of these approaches can deal with all types of errors, and all of these methods assumed that probes are all unique. L&H modified Hsu's algorithm and devised an iterative clustering algorithm for the physical mapping problem [18]. Their method could handle the four types of errors. However, the performance goes down dramatically when the error rate is increased. Our GAPM adopts a two stage genetic algorithm with a much better performance.

Furthermore, with a slightly change, GAPM could be also used to solve the problem of sequence assembly since the 0-1 matrix of the overlapping relationships of the fragments should also satisfy the COP. In this paper, we only demonstrate the performance of GAPM on the physical mapping problem. The application of GAPM for the sequence assembly problem will be the topic of another paper.

2 Methods

The workflow of GAPM consists of the following steps: error detection, a two stage genetic algorithm, and a post-processing step. The error detection step tries to detect three error types, excluding false negatives. After the error treatment, we generate pseudo clones by merging overlapped clones for recovering the false negatives. At the first stage of the genetic algorithm, we use GA1 to generate a partial probe ordering for each pseudo clone, and to detect false negatives within each clone. Then we generate the probe neighborhood information from multiple runs of GA1, one run for each pseudo clone. This information is used to construct the whole probe ordering using GA2 at the second stage. Finally, we analyze and report the probe ordering in the post-processing step. The details of each step are described in the following sub-sections.

2.1 Error Detection and Treatment

This step first detects non-unique probes, then false positives, followed by chimeric clones. These detected errors will be treated. Since false negatives can be detected only after a probe ordering is given, instead of detecting false negatives, we construct a pseudo clone for each clone after the above error treatment. Each pseudo clone can be treated as a cluster of probes and is used as input for GA1. The details are described in the following subsections.

The Detection of Non-unique Probes The frequencies of non-unique probes are very likely higher than those of unique probes since their sequences appear more than once within the whole chromosome. Hence, we use the frequencies of probes to detect the non-unique probes.

We calculate the frequency of each probe in all clones. Then for each clone, we determine locally whether there is a non-unique probe by comparing the frequency of each probe in the clone with the clone's average frequency, which is calculated from all probe frequencies in the clone excluding the smallest and the highest frequencies. If the frequency of a probe is higher than a threshold, currently defined as 1.33 times of the average frequency, the probe is suspected to be a non-unique probe. Each suspect probe P_j within C_i will be removed from C_i. The above procedure will be repeated until no more suspect probes can be found.

Note that, some unique probes may be falsely suspected and removed due to the effect of multiple errors, thereby possibly generating more false negatives. These cases can be remedied in the procedure of false negative detection. On the other hand, some cases cannot be easily detected due to combined errors, and these hard cases can be regarded as false positives, which are then handled in false positive detection.

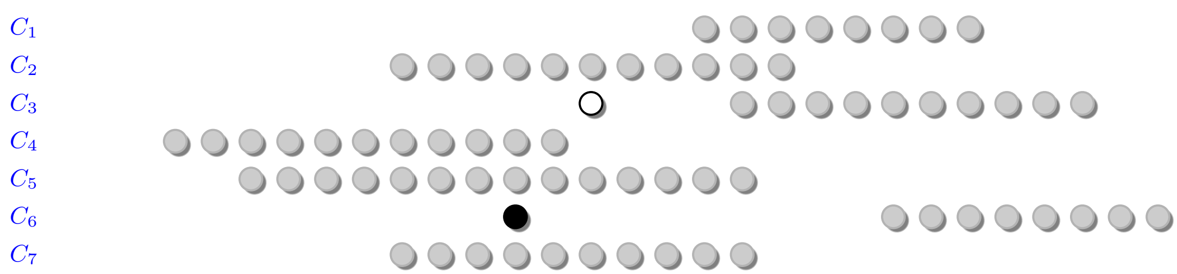

Fig. 1. Two possible false positives shown in white and block circles. The probes attached to C_i are usually attached to several common clones. In contrast, a false positive probe attached to C_i is attached to some clones that do not belong to these common clones. For example, the black probe is a highly probable false positive attached to C_6.

The Detection of False Positives The probes attached to C_i are usually attached to several common clones. However, a false positive probe attached to C_i, could be attached to some clones that do not belong to these common clones. For example, consider two probes represented by white and black circles in Figure 1 and determine which is more likely a false positive. The white probe, i.e., the probe represented by the white circle, is not too far from a true positive and attached to some common clones as some probes in C_3. And, the black probe is farther from a true positive and attached to different clones from those that other probes in C_6. Comparing these two probes, the black probe is a highly probable false positive attached to C_6. Motivated from this observation, we define *distant clone* with respect to a given clone C_i and use the number of distant clones to determine whether a probe P_j in clone C_i is a false positive.

A clone $C_{i'}$ is a clone *distant from* C_i if the number of probes attached to both clones is less than 3. A false positive probe P_j with respect to C_i will likely be attached to different clones that other true positive probes are not. In other words, most clones that P_j attached to are *distant from* C_i. Specifically, a probe

P_j is suspected to be a false positive with respect to C_i if more than a portion f of the number of clones containing P_j are *distant from* C_i. The threshold f is set to be 0.65. Once a false positive is detected, remove P_j from C_i. Sequentially, we process all probes attached to C_i. The above procedure is applied to all clones.

The Detection of Chimerical Clones To detect whether a clone C_i is chimeric, we check whether all of the probes attached to C_i form at least two disjoint sets such that probes in different sets do not attach to any common clone other than C_i. Any two distinct probes P_j and $P_{j'}$ attached to C_i are called strong neighbors if they are attached to at least one common clone other than C_i. We use this strong neighbor relation to connect probes into a probe set and determine whether all probes attached to C_i form only one set.

For each clone C_i, randomly start with a probe P_j to form a probe set S. Then expand S to include one more probe $P_{j'}$ from the remaining probes not in S that is a strong neighbor of any probe in S. Repeat the expansion of S until no more probes can be included. If the resulting set S does not contain all probes attached to C_i, then we consider C_i as a chimeric clone and repeat the above procedure to construct other probe sets S' for probes not in S to exhaust all probes attached to C_i.

Once C_i is detected as a chimeric clone, we choose the largest probe set S and replace C_i with a shorter clone $C_{i'}$ with probes in S. Note that some chimeric clones cannot be easily detected when their composing clones are close in distance. These cases will be treated as regular clones, and the gaps between their composing clones will be considered as false negatives in our probe ordering determination stage. Furthermore, we use clones with s_1 to s_2 probes in later probe ordering procedure. The range between s_1 and s_2 represents the regular sizes of clones. The size bound s_2 can help remove some chimeric clones that cannot be detected by the procedure.

Pseudo Clones Since we cannot determine whether there is a false negative within a clone before the true probe ordering is given, false negatives are harder to be detected than the other three types of errors. Furthermore, arranging the probe ordering for each clone one by one may generate ambiguity in some probe's ordering due to false negatives. For example, the best probe ordering for clone A is [Probe_1, Probe_3, Probe_4], and that for clone B is [Probe_1, Probe_2, Probe_4]. Each of the two clones contains a false negative. Two possible orders of these probes are [Probe_1, Probe_2, Probe_3, Probe_4] and [Probe_1, Probe_3, Probe_2, Probe_4]. In order to reduce the ambiguity in probe ordering caused by false negatives, we generate *Pseudo Clones* to replace the original clones for the subsequent genetic algorithm.

A *pseudo clone* is a synthetic clone which is extended from a specific clone and formed by a union of several overlapped clones, i.e., clones sharing some common probes. For each clone C_i, we generate a pseudo clone PC_i by the following procedure. Initially, PC_i is simply C_i. Then we extend PC_i by merging the neighboring clone $C_{i'}$, which has most probes in common with any clone in PC_i. Repeat the pseudo clone extension until all neighboring clones have been exhausted or the pseudo clone is hybridized with s_2 probes since we only consider clones containing at most s_2 probes. Figure 2 shows an example of a pseudo clone which is generated from the clones below. Each row of white circles represents a clone with or without false negatives. The row with black circles represents a pseudo clone which is the union of the entire clones. By merging the probe presence into a pseudo clone and arranging the probe ordering for the pseudo clone rather than for the original ones, we can greatly reduce the ambiguity. Note that after constructing pseudo clones from all clones, some pseudo clones are possibly identical, which are removed to perform subsequent genetic algorithms.

2.2 Two Stage Genetic Algorithms to Generate Probe Ordering

In order to obtain a good ordering of probes, we use two stage genetic algorithms. The first stage genetic algorithm, GA1, uses pseudo clones to generate some good partial orders of probes which serve as constraints on the probe connection for the second stage genetic algorithm. The second stage genetic algorithm, GA2, aims to generate a good global ordering of probes.

The First Stage of the GAPM(GA1) For each pseudo clone PC_i, GA1 aims to generate the best probe ordering for the probes contained in PC_i based on a defined fitness function. Note that if there are m' pseudo clones ($m' \ll m$, m: the number of original clones), we perform m' times of GA1 to generate a collection of partial orders of probes. A genetic algorithm usually comprises the following steps: chromosome initialization, reproduction, crossover, mutation operation and uses a fitness function to determine a good solution. Details of GA1 are described below.

Chromosome Initialization Each chromosome in GA1 represents a possible probe order. To expedite the execution of GA1, instead of randomly initializing a chromosome as in most genetic algorithms we initialize the chromosome from a possibly good probe order.

In general, the size of *CloneList* $(P_j, P_{j'})$, which is defined as the set of clones containing both P_j and $P_{j'}$, is related to the distance between the two probes in the ordering. The closer the two probes are, the more common clones they may have. We define the probability for probes, P_j and $P_{j'}$ to be adjacent, denoted by $Adj(P_j, P_{j'})$, as follows:

$$Adj(P_j, P_{j'}) = \frac{|\ CloneList(P_j, P_{j'})\ |^3}{\sum_{j''} |\ CloneList(P_j, P_{j''})\ |^3}$$

To initialize the chromosome, we randomly select a probe chromosome P_j as the first locus of the chromosome. Given the probe P_j located at the previous locus, an unselected probe $P_{j'}$ is chosen according to $Adj(P_j, P_{j'})$ and assigned to the next locus. The assignment continues until all l probes have been chosen and assigned on the chromosome.

Fitness Function GA1 evaluates the fitness score of each chromosome using a fitness function. Given an ordering of l probes represented by a chromosome, we generate a $k \times l$ 0-1 matrix M with k clones overlapping with the psudo clone PC_i as rows and columns arranged according to the probe order. We use $M[i]$ to denote the i-th row of M. The fitness function is defined on M, denoted by $f(M)$, which depends on the length of consecutive ones in the matrix, as an aggregation of two fitness subfunctions: f_1 and f_2. The function f_1 evaluates the fitness score before the detection of false negatives. Since false negatives can be determined given the probe order, the function f_2 evaluates fitness score after the detection of false negatives.

The fitness function $f_1(M)$ evaluates the lengths of consecutive ones in M subtracted by incurred penalties. For each each $M[i]$ with r consecutive one segments, its *ConsecutiveOnes* score is given by $\sum_{p=1}^{r} \sum_{q=1}^{l_p} q$, where l_p denotes the length of segment p. For example, the row (11101101) has three consecutive-one segments having length 3, 2 and 1, respectively. The row's *ConsecutiveOnes* score is 10 (= 6+3+1). Then *ConsecutiveOnes*(M) is the sum of the *ConsecutiveOnes* scores of all rows in M. Since each clone is in size between s_1 and s_2, rows in M with the numbers of 1's less than s_1 should not be fully included by the pseudo clone PC_i. Based on that fact, we give a penalty for those rows with the number of 1's less than s_1 which are fully included by PC_i. We say a row is fully included which means none of the 1's located at either of the two ends. The penalty function, called *IslandPenalty* is defined as follows. *IslandPenalty*$(M[i])$ $= [s_1 - NumOnes(M[i])]^2 \times d_i^2$, where *NumOnes* returns the number of 1's in the $M[i]$ and d_i is the shortest distance of the 1's to either of the two ends of $M[i]$. Note that *NumOnes* always returns s_1 if the number of 1's is greater than s_1. For example, given s_1= 4, the row (0110100) has *NumOnes* = 3 and d = 1 since the leftmost 1 is at distance 1 from the left-end and the rightmost 1 is at distance 2 from the right-end. The *IslandPenalty* of that row is 1. Therefore the fitness function $f_1(M)$ is defined as below.

$$f_1(M) = ConsecutiveOnes(M) - \sum_{i=1}^{k} IslandPenalty(M[i])$$

While f_1 does not consider the false negatives, function f_2 does. To detect false negatives, we use a sliding window of 3 columns to form a $k \times 3$ submatrix of M. For each of the $k \times 3$ submatrix, if among these k rows the number of 111 rows is greater than the number of 101 rows by at least a threshold t, we consider that the 101 rows likely have a false negative and then replace these rows by 111. Given the threshold $t = 6$, Figure 3 shows an example of false negative. In the indicated $k \times 3$ submatrix, there are seven 111 rows and only one 101 row. Thus a false negative at the 101 row is detected and the middle entry of the row is replaced by 1. Similarly, we detect two successive false negatives in $k \times 4$ submatrix of M by comparing the numbers of 1111 rows and 1001 rows.

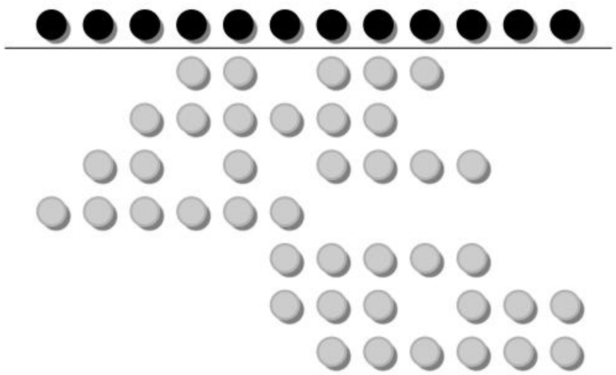

Fig. 2. An example of a pseudo clone which is generated from the clones below, where the first clone is the initial seed to generate the pseudo clone.

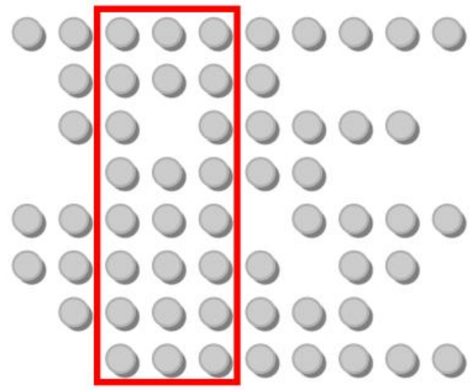

Fig. 3. Detecting false negatives by screening the difference between 111's and 101's within a sliding window.

Let M' represent the resultant $k \times l$ matrix after the false negative recovery. The fitness function f_2 is defined on M'. Since the recoveries of false negatives increase $ConsecutiveOnes(M')$, in order to prevent the population in the evolutionary procedure from generating many false recoveries, we define a penalty for such recoveries, called RecoveryPenalty, as follows: $RecoveryPenalty(M') = 5 \times \left(\sum_{r=1}^{fnr1} r + \sum_{r'=1}^{3 \times fnr2} r' \right)$, where *fnr1* and *fnr2* are the numbers of false negative recoveries for 101 and 1001, respectively. The penalty increases more rapidly when the number of recoveries is getting larger. We also consider another penalty in f_2, called DistancePenalty, which incur to rows having scattered 1's. We define $DistancePenalty = 5 \times \sum_{i=1}^{k} ProbeDistanceSum(M'[i])$, in which the function *ProbeDistanceSum* calculates the sum of distances between any pair of 1's which are separated by 0's. For example, the row (1001100101) has the *ProbeDistanceSum* of 5 (=2+2+1). Finally, f_2 is defined as follows: .

$$f_2(M) = ConsecutiveOnes(M') - DistancePenalty(M') - RecoveryPenalty(M')$$

When the score of $f_2(M')$ is negative, it is reset to be 0. In the first few generations, f_2 usually returns a negative score; as probe ordering is improving in the evolution, a chromosome will likely get a bonus from f_2.

Reproduction Operation The population for the next generation is generated as follows. After ranking all chromosomes by fitness scores, the top half of population are kept for the next generation. The other half are discarded and we generate new chromosomes using either the crossover operation or the chromosome initialization, depending on the CrossoverRate, which denotes the proportion of new chromosomes produced by the crossover operation. For example, if the CrossoverRate is 0.7, then 70% of new chromosomes are produced by the crossover operation, and 30% are produced by the chromosome initialization.

Chromosomes in the first half of population serve as parent candidates in the crossover operation. We randomly select two distinct candidates, say $parent_1$ and $parent_2$, and a cut point cp to produce an offspring ch. To compose ch, the probe arrangement before the cp is copied from $parent_1$ and is sequentially extended by probes in $parent_2$ that have not yet been included in ch. Figure 4 shows an example of an offspring produced by the crossover operation.

The mutation operation in GA provides the population a reasonable diversity and prevents the offspring from resembling their parents. The MutationRate is a parameter which denotes the probability of a locus being mutated. For example, if the MutationRate is 0.005, then each locus in a chromosome has a probability 0.5% to be mutated. If a locus is going to be mutated, another locus is randomly selected and we exchange the probes of the two loci.

Probe Adjacency Lists Generated from the GA1 results GA1 would return one or more probe orders with the highest fitness score for each pseudo clone. We do not use GA1's results of different pseudo clones directly since pseudo clones may overlap in some probes and these overlapping probes may have inconsistent ordering in different runs of GA1. Therefore we generate a probe adjacency list, denoted as $AdjacentList[P_j]$, for each probe P_j. A probe $P_{j'}$ is listed in the $AdjacentList[P_j]$ if P_j and $P_{j'}$ are adjacent in any one of the returned probe orders and $CloneList$ $(P_j, P_{j'})$ is not empty. The adjacency relation is symmetric which means if $P_{j'}$ is in the $AdjacentList[P_j]$ then the P_j is also in the $AdjacentList[P_{j'}]$.

The best case is that each probe only has two adjacent probes excluding the head and the tail probes. However, the false negatives and the probe ordering inconsistency would result in long adjacency lists for some probes. We use GA2 to generate the whole probe ordering according to those adjacency lists.

2.3 The Second Stage of the GAPM (GA2)

Most operations of GA2 are similar to those of GA1 except the *chromosome initialization*, *mutation* and *fitness function*. We describe the details of these operations in the following sub-sections.

Chromosome Initialization In order to expedite the termination of GA2, we generate contigs, which are a set of probes arranged in a specific order such that two consecutive probes are adjacent which means one can be found in *AdjacentList* of the other to initialize a chromosome. Each probe which has only one adjacent probe is in turn used to initiate a contig. If no such probe can be found, i.e., all probes have at least two adjacent probes, all probes could be used to initiate contigs. Let P_j be selected as the head probe to initiate a contig. We extend the contig from P_j by randomly selecting an unused probe $P_{j'}$ from the $AdjacentList[P_j]$ and then we use $P_{j'}$ for further extension. The extension continues until no more unused probe can be found from the adjacent list of the last probe to be included into the contig. Another contig is then initiated if there are other unused probes. Finally a chromosome is completed with one or more contigs to exhaust all probes.

Fitness Function Given a probe ordering PR with n probes, we generate the corresponding 0-1 matrix $M^{'}$. The fitness function f_3 defined on PR considers not only the resulting *ConsecutiveOnes* score but also the score of probe adjacency in the given probe ordering and is given as follows:

cut point

$parent_1$	1	3	2	4	5	7	6
$parent_2$	2	5	6	1	3	4	7
offspring	1	3	2	4	5	6	7

Fig. 4. An example showing how a offspring being produced from two parents using single cut point in the crossover operation.

$$f_3(PR) = ConsecutiveOnes(M^{'}) + \sum_{s_1=0,i=2}^{i=n} s_i, \begin{cases} s_i = s_{i-1}+1, & \text{if } PR[i-1] \leftrightarrow PR[i] \\ s_i = 0, & \text{otherwise} \end{cases}$$

where the symbol $\leftrightarrow$ represents the adjacency relation.

Mutation Operation Given a chromosome *ch*, let *mch* denote the mutant of *ch*. Initially, all the loci of *mch* are undefined and then determined as follows: The mutation operation starts with the first locus, i.e., *locus* is 1. If the $ch[locus]$ is going to be mutated, select a probe not yet included in *mch* to assign to $mch[locus]$. Then a new contig is generated beginning with $mch[locus]$ to include into *mch*. The next locus in *ch* to consider for mutation is the one next to the last probe of the newly generated contig. If $ch[locus]$ does not mutate and has not yet included in *mch*, then $ch[locus]$ is directly assigned to $mch[locus]$. If $ch[locus]$ has been included in *mch*, we arbitrarily choose a probe not yet included in *mch* to initiate a new contig. We repeat the operation until all probes are assigned in *mch*.

2.4 Post-processing

GA2 will return one or more probe orders with the highest fitness score. The probe orders may consist of several contigs due to either the probe deletions in error treatment or the imperfect probe arrangement. However, if some marginal probes of two contigs may actually coexist in the same clone before the error treatment, we then connect these two contigs.

Only the contigs with at least 10 probes are considered for possible connection. For any pair of contigs, say *Contig1* and *Contig2*, we check their possible connections *Contig12* and *Contig21*, which means *Contig1* followed by *Contig2* and *Contig2* followed by *Contig1*, respectively, by confidence scores defined as follows. Specifically, to determine whether *Contig12* can be formed, we check whether some clones C_i (the original clones before error detection and treatment) contain some probes from *TailProbeSet1*, i.e., the last five probes of *Contig1*, and *HeadProbeSet2*, i.e., the first five probes of *Contig2*. If a clone C_i contains n_1 probes in *TailProbeSet1* and n_2 probes in *HeadProbeSet2*, we assign a confidence score of $\min(n_1, n_2)$ to *Contig12*. The confidence score of forming *Contig12*, denoted by *Score*(*Contig12*), is defined as the sum of confidence scores of all such C_i's. Similarly, we can calculate confidence score of forming *Contig21*, denoted by *Score*(*Contig21*), with respect to the last five probes of *Contig2* and the first five probes of *Contig1*.

If *Score*(*Contig12*),and *Score*(*Contig21*) differ by at least 5, we connect the two contigs with higher score; otherwise, we leave them unconnected. We repeat the contig connection procedure until no more contigs can be connected. Then the resulting contigs and the contigs with 3-9 probes from GA2 output are reported as the final solution for probe ordering.

3 Results

GAPM is developed as a parallel computing program under Linux environment. It is implemented using the C++ and MPICH library. The parameters are as follows: the population size is 800, the number of generations is 600, CrossoverRate is 0.7 and MutationRate is 0.005. Since it is difficult to obtain abundant wet-lab data, we conduct experiments on the synthetic data for evaluating the performance of GAPM and compare with L&H's method using the same dataset.

3.1 Dataset

To generate a synthetic data, we simulate the process of restriction enzyme digestion on a DNA sequence. A DNA sequence is a string of n 1's, each representing a probe, and we can assume without loss of generality that the correct probe ordering is 1, 2, ...,n. We randomly cut the sequence into several fragments of size of 5 to 15, called clones. We repeat the digestions using more duplicate DNA sequences until there are m clones. Then we generate an $m \times n$ 0-1 matrix and alter the entries randomly to simulate the four types of errors.

We follow L&H's design to generate noisy data. The errors of false positives and false negatives are at three different levels, 3%, 5% and 10%. Within each error rate, the ratio of false positives to false negatives is set to be 1 to 4. For example, let the total number of 1's in a 100×100 matrix be k. We will generate $0.02k$ false positives by randomly changing the same number of 0's to 1's if the error rate is 10%. Similarly, we will generate $0.08k$ false negatives by randomly changing the same number of 1's to 0's. We also randomly generate additional 2% chimeric clones ($0.02m$) and 2% non-unique probes ($0.02n$) for each matrix. To generate a chimeric clone, we make a copy of the 1's of a row to the other by randomly selecting two different rows. To generate a non-unique probe, we make a copy of 1's of a column to the other by randomly selecting two different columns. We generate 1000 matrices of sizes 100×100, 200×200, and 400×400 for each false-positive and false-negative error rate given the same error rates for the other two errors, respectively (i.e., 9000 matrices totally).

3.2 Performance Evaluation

We also follow the evaluation method proposed by L&H. For a probe v, let d_1 be the number of probes ordered to the left of v but whose indices are greater than v, and d_2, the number of probes ordered to the right of v whose indices are less than v. Let the displacement $d(v)$ be the larger of d_1 and d_2 of probe v.

The *displacement* $d(v)$ gives an approximate measure of the distance of probe v from its correct position. L&H proposed the following three measures for estimating the total deviation of the resulting probe orders:

1. The *average displacement* of a probe ordering is the average of the displacement of all probes.
2. If the displacement of a probe v is more than 4, we say v is a jump probe. The *jump percentage* is the number of jump probes divided by the total number of probes.
3. The *average difference* of a probe ordering is the average of the difference in the probe indices of adjacent probes.

For example, given the probe ordering shown in Figure 5, $d(2) = 6$ (there are 6 probes ordered to the left of probe 2 whose indices are greater than 2), $d(6) = 1$, and $d(8) = 6$ (there are 6 probes ordered to the right of probe 8 whose indices are less than 8). Thus, both probe 2 and probe 8 are jump probes. The average displacement is 1.7, and the average difference is 3.2.

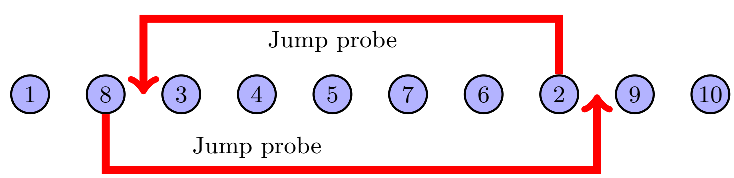

Fig. 5. Jump probes.

3.3 Experiment Results

We evaluated the performance of GAPM and L&H method using the above three measures. Each experiment result is evaluated from 1000 different cases of different matrix sizes and error rates. (Since all test sets have the same error rates for chimeric clones and non-unique probes, henceforth say different error rates simply mean different rates of false positives and false negatives.) The jump percentage of GAPM and L&H on different test sets are shown in Figure 6, in which the x-axis represents different test sets, e.g., M100_ER10% represents the test sets in size of 100×100 and 10% of error rate. The results show that GAPM has much smaller jump percentage than L&H for different test sets. For example, the average jump percentage of GAPM is 0.15% for M400_ER10%, which means there is around 0.6 jump probe in average for arranging 400 probes with 10% of error rate. Most of probes are arranged in or near their correct positions. However, the average jump percentage of L&H for the same test sets is 4.17%. It implies there are 16.68 probes in average being arranged far from their correct positions.

Average displacements of GAPM and L&H on different test sets are shown in Figure 7. GAPM generates probe orders with smaller displacement than L&H. For example, the average displacement of GAPM is 0.41 in M400_ER10%, whereas that of L&H is 0.83. The smaller the displacement, the closer each probe is to its correct position. The comparison results imply that the probe orders reported by GAPM are more reliable than those by L&H.

Average differences of GAPM and L&H on different test sets are reported in Figure 8. The measure estimates the difference in the probe indices of adjacent probes. GAPM generates probe orders with smaller difference than L&H, whereas a correct probe ordering gives an average difference of 1. For example, the average difference of GAPM is 1.51 for the matrices in M400_ER10%, better than 2.1 reported by L&H. It implies that more probes are arranged together with their nearest neighbors by GAPM.

L&H method removes probes during the process of probe ordering according to their rules. In contrast, GAPM removes probes during error detection and treatment and probes in the contigs generated by GA2 with at most two probes. Figure 9 shows the comparison results of the percentage of deleted probes between GAPM and L&H. GAPM deleted much less probes. Furthermore, the number of deleted probes is relatively stable for different error rate. In contrast, the number of deleted probes by L&H increases when the error rate is increased, i.e., there is a positive correlation between the percentage of deleted probes and the error rate in the results of L&H. For example, the average percentage of deleted probes in M100_ER03% of L&H is 12.69%, and it increases to22.33% and 48.54% when the error rate is increased to 5% and 10%, respectively. Noteworthy, those of GAPM are 1.88%, 1.89% and 2.01%, respectively. It strongly implies that GAPM is less sensitive to the error rate.

The final probe orders may consist of several contigs due to either the probe deletions in error treatment or the imperfect probe arrangement, whereas the correct probe ordering contains only one contig. Therefore the number of contigs is also an important measure for evaluating the performance. Figure 10 shows the comparison results between GAPM and L&H in terms of the average number of contigs. L&H generates more contigs than GAPM. It could be easily observed that there is a positive correlation between the number of contigs and the number of probes in the results of L&H. For example, the average numbers of contigs generated by L&H using the matrices in sizes of 100×100, 200×200 and 400×400 with 3% of error rate are 2.27, 3.93 and 7.00 respectively. However, those by GAPM are 1.01, 1.04, and 1.11 respectively. It strongly implies that GAPM is relatively robust when the size of matrices increases.

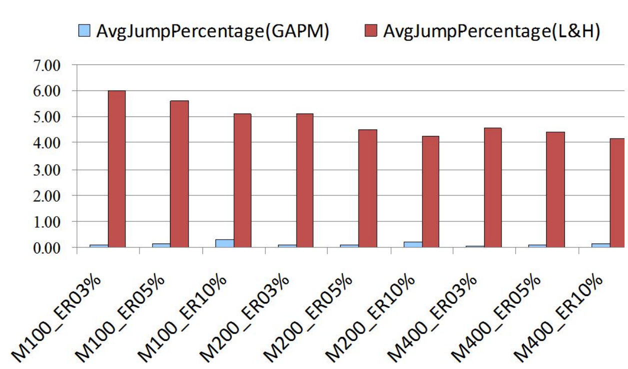

Fig. 6. The comparison of jump percentage.

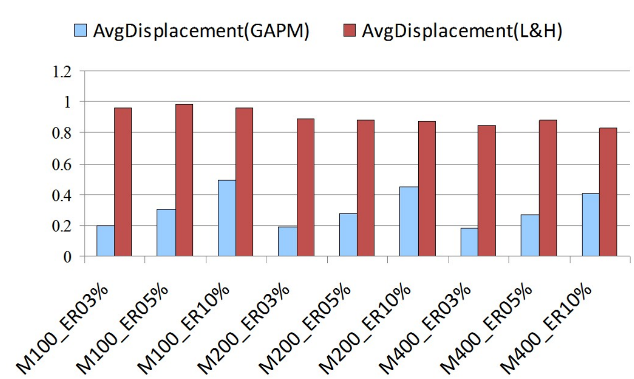

Fig. 7. The comparison of average displacements.

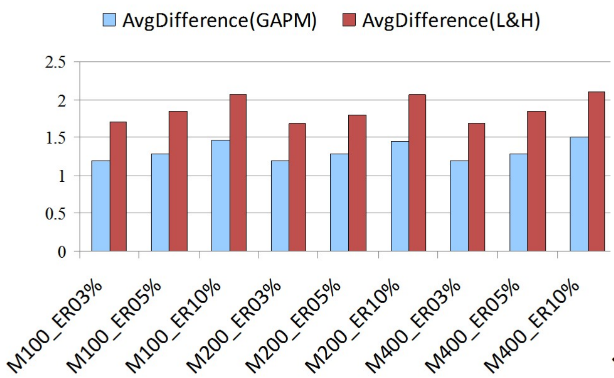

Fig. 8. The comparison of average difference.

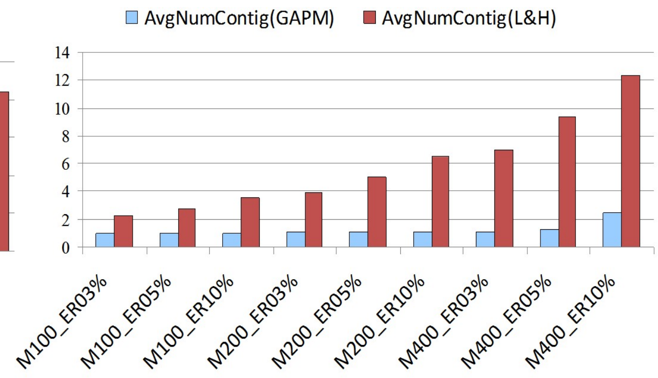

Fig. 9. The comparison of average number of deleted probes.

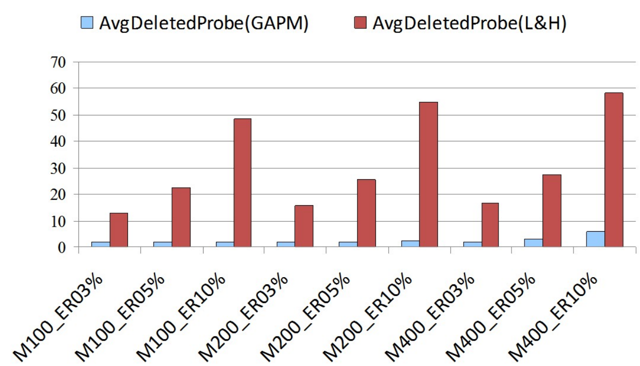

Fig. 10. The comparison of average number of contigs.

4 Conclusions

In this paper, we propose a novel approach for constructing physical maps. Since different types of error are inevitable in datasets, we use the cross reference of probe-clone relationships for error detection without complicated rules. As the experiment result shows, most errors can be removed. After the initial error reduction, we use a two stage genetic algorithm which greatly reduces the search space from $n!$ possible probe orderings to a much smaller one. Specifically, we cluster the probes into several groups and each group contains at most 15 probes. Then using each group of probes, we form pseudo clones, which aim to dilute the effect of false negatives. Then GA1 uses each pseudo clone as input for chromosome initialization and subsequent procedures. The small size of a pseudo clone makes it easier to find a good probe ordering in GA1. Noteworthy, the fitness function used in GA1 is an aggregation of two functions, which takes into consideration of not only the COP but also the effect of false negatives. The fitness function helps the population gradually improve during the evolution. After a good ordering for each pseudo clone is determined by GA1, we generate the probe neighborhood information, which provides the probe ordering constraint for GA2 and then determine an overall good probe ordering.

We simulate the process of restriction enzyme digestion on DNA sequences and generate a huge amount of synthetic data for evaluating the performance of GAPM. By introducing those four types of errors, GAPM could still generate satisfactory probe orderings. According to the experiment results, GAPM is more reliable and robust than the L&H's method. It is also less sensitive to the error rate and the size of matrices. The physical map generated could be a reference sequence for anchoring the short DNA sequence fragments produced by the next generation sequencing technologies.

Because GAPM is very modular, it can be easily modified. The fitness function is one example. The effect of different designs of the fitness function could be easily checked through the evolution of the population. In fact, any problem which could be modeled as an optimization problem on 0-1 matrices satisfying the COP could be a potential application of GAPM.

References

1. Pop, M., Salzberg, S.L.: Bioinformatics challenges of new sequencing technology. Trends in Genetics **24**(3) (2008) 142–149
2. Cristian Coarfa, A.M.: Pash 2.0: Scaleable sequence anchoring for next-generation sequencing technologies. In: Pacific Symposium on Biocomputing. (2008) 102–113
3. Kim, H., Hurwitz, B., Yu, Y., Collura, K., Gill, N., SanMiguel, P., Mullikin, J.C., Maher, C., Nelson, W., Wissotski, M., Braidotti, M., Kudrna, D., Goicoechea, J.L., Stein, L., Ware, D., Jackson, S.A., Soderlund, C., Wing, R.A.: Construction, alignment and analysis of twelve framework physical maps that represent the ten genome types of the genus oryza. Genome Biology **9**(2) (2008)
4. Margulies, M., Egholm, M., Altman, W.E., Attiya, S., Bader, J.S., Bemben, L.A., Berka, J., Braverman, M.S., Chen, Y.J., Chen, Z., Dewell, S.B., Du, L., Fierro, J.M., Gomes, X.V., Godwin, B.C., He, W., Helgesen, S., Ho, C.H., Irzyk, G.P., Jando, S.C., Alenquer, M.L.I., Jarvie, T.P., Jirage, K.B., Kim, J.B., Knight, J.R., Lanza, J.R., Leamon, J.H., Lefkowitz, S.M., Lei, M., Li, J., Lohman, K.L., Lu, H., Makhijani, V.B., Mcdade, K.E., Mckenna, M.P., Myers, E.W., Nickerson, E., Nobile, J.R., Plant, R., Puc, B.P., Ronan, M.T., Roth, G.T., Sarkis, G.J., Simons, J.F., Simpson, J.W., Srinivasan, M., Tartaro, K.R., Tomasz, A., Vogt, K.A., Volkmer, G.A., Wang, S.H., Wang, Y., Weiner, M.P., Yu, P., Begley, R.F., Rothberg, J.M.: Genome sequencing in open microfabricated high density picoliter reactors. Nature **437** (2005) 376 – 380
5. Booth, K.S., Lueker, G.S.: Testing for consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. Journal of Computer and System Sciences **13**(3) (1976) 335–379
6. Hsu, W.L.: A simple test for the consecutive ones property. Journal of Algorithms **43**(1) (2002) 1–16
7. Karp, R.M.: Mapping the genome: some combinatorial problems arising in molecular biology. STOC '93: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing (1993) 278–285
8. Backofen, R., Gilbert, D.: Bioinformatics and constraints. Constraints **6**(2) (2001) 141–156
9. Christof, T., Jünger, M., Kececioglu, J., Mutzel, P., Reinelt, G.: A branch-and-cut approach to physical mapping with end-probes. In: RECOMB '97: Proceedings of the First Annual International Conference on Computational Molecular Biology, New York, NY, USA, ACM (1997) 84–92
10. Golumbic, M.C., Kaplan, H., Shamir, R.: On the complexity of dna physical mapping. Advances in Applied Mathematics **15**(3) (1994) 251–261
11. Cuticchia, A.J., Arnold, J., Timberlake, W.E.: Ods - ordering dna-sequences - a physical mapping algorithm based on simulated annealing. Computer Applications in the Biosciences **9**(2) (1993) 215–219
12. Alizadeh, F., Karp, R.M., Weisser, D.K., Zweig, G.: Physical mapping of chromosomes using unique probes. SODA '94: Proceedings of the fifth annual ACM-SIAM Symposium on Discrete Algorithms (1994) 489–500
13. Jain, M., Myers, E.W.: Algorithms for computing and integrating physical maps using unique probes. Journal of Computational Biology **4**(4) (1997) 449–466
14. Mayraz, G., Shamir, R.: Construction of physical maps from oligonucleotide fingerprints data. Journal of Computational Biology **6**(2) (1999) 237–252
15. Goncalves, L., Ze-Ze, L., Pinheiro, H.P., Amaral-Turkman, M.A.: Statistical aspects in physical mapping application to the genome of o. oeni strain gm. Biometrics **61**(2) (2005) 481–487
16. Ukkonen, A., Fortelius, M., Mannila, H.: Finding partial orders from unordered 0-1 data. KDD '05: Proceeding of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (2005) 285–293
17. Grigoriev, A., Mott, R., Lehrach, H.: An algorithm to detect chimeric clones and random noise in genomic mapping. Genomics **22**(2) (1994) 482–486
18. Lu, W.F., Hsu, W.L.: A test for the consecutive ones property on noisy data - application to physical mapping and sequence assembly. Journal of Computational Biology **10**(5) (2003) 709–735