Hindawi Publishing Corporation
Applied Computational Intelligence and Soft Computing
Volume 2012, Article ID 945401, 11 pages
doi:10.1155/2012/945401

Research Article

An Efficient Genome Fragment Assembling Using GA with Neighborhood Aware Fitness Function

Satoko Kikuchi and Goutam Chakraborty

Faculty of Software and Information Science, Iwate Prefectural University, Takizawa-mura, Iwate 020-0193, Japan

Correspondence should be addressed to Goutam Chakraborty, goutam@iwate-pu.ac.jp

Received 12 March 2012; Accepted 11 May 2012

Academic Editor: Qiangfu Zhao

To decode a long genome sequence, shotgun sequencing is the state-of-the-art technique. It needs to properly sequence a very large number, sometimes as large as millions, of short partially readable strings (fragments). Arranging those fragments in correct sequence is known as fragment assembling, which is an NP-problem. Presently used methods require enormous computational cost. In this work, we have shown how our modified genetic algorithm (GA) could solve this problem efficiently. In the proposed GA, the length of the chromosome, which represents the volume of the search space, is reduced with advancing generations, and thereby improves search efficiency. We also introduced a greedy mutation, by swapping nearby fragments using some heuristics, to improve the fitness of chromosomes. We compared results with Parsons' algorithm which is based on GA too. We used fragments with partial reads on both sides, mimicking fragments in real genome assembling process. In Parsons' work base-pair array of the whole fragment is known. Even then, we could obtain much better results, and we succeeded in restructuring contigs covering 100% of the genome sequences.

1. Introduction

1.1. What Is Genome? The study of bioinformatics is one of the most vibrant area of research, whose important applications are growing exponentially. A good starting point is the introductory book by Neil and Pavel [1]. Varieties of interesting applications are reported, where computational algorithms play an important role, as [2, 3]. Of all bioinformatics researches, genome sequencing received high importance from the beginning of this century, starting with human genome [4], to sequencing of crops [5].

Genome is the complete genetic sequence made from an alphabet of four elements, Adenine (A), Thymine (T), Cytosine (C), and Guanine (G). The letters A, T, C, G represent molecules called nucleotides or bases. In a living cell, it appears in a double helix structure [6]. Every base A in one strand is paired with a T on the other strand, and every base C is similarly paired with G. These pairs are called base-pairs, or simply bp [7, 8].

The genome (DNA) sequences are enormously long from a few thousand nucleotides for small viruses to more than 3 giga nucleotides for human. Genomes like that of wheat (1.7×10^{10} bp) and lily (1.2×10^{11} bp) are longer than human genome (from NCBI database (the National Center for Biotechnology Information—http://www.ncbi.nlm.nih.gov/)). Obviously, deciphering them, though important, is very complex.

1.2. Why Is It Important to Decipher DNA Sequence? DNA sequence, responsible for producing different proteins, is at the root of functioning of a living organism. Decoding genome sequence is thus the first step to understand the function as well as malfunction of living things, for medical, agricultural, and many other research areas. Investigation of human genome could lead to the cause of inherited diseases and the development of medical treatments for various illnesses. The genome analysis is useful for the breeding of improved crops. Moreover, it is the key information for investigations in evolutionary biology. It is hoped that the platypus genome, which is very recently decoded in 2008, would provide a valuable resource for in-depth comparative analysis of mammals [9].

1.3. Existing Fragment Assembly Methods. Sanger sequencing [10] method is well known and most commonly used for reading genome sequences. It uses *Gel electrophoresis* that facilitates reading the bases due to reaction of the fluorescent dye staining different bases differently. But it is possible to read only a few hundred base-pairs. To overcome this problem, the target genome is cloned into copies and cut into small *fragments*. Base sequences of those fragments are read and the original genome is reassembled using information of the overlapping portions of the fragments. This procedure is called *shotgun sequencing* and is the most commonly used method for genome sequencing [11–13]. In fact, Lander et al. [4], using shotgun sequencing, presented an initial sequencing of human genome of ≈3.5 Gbps length by 2001.

Most of the existing fragment assembly systems read the fragment base-sequence by Sanger technique and reconstruct the original genome sequence with their proprietary assembling algorithms. Many assembling algorithms were proposed, the important ones being TIGR assembler [14], RAMEN [15], Celera Assembler [16], CAP3 [17], and Phrap [18]. To reconstruct the target genome, lots of fragments are needed and assembling those into correct sequence is an NP-hard problem.

1.4. GA-Based Genome Fragment Assembling Works. The present work is based on genetic algorithm (GA), which is modified to be efficient for such array assembling problem. During last ten years a few works were reported to use genetic algorithm or similar techniques like *clustering algorithm* and *pattern matching algorithm*, to solve fragment assembling problem. A survey with comparison of their respective performances is reported by Li and Khuri [19]. Most of the GA based works are simple modifications of Parsons et al.'s works [20]. Recent works [21, 22] used distributed GA. Fang et al.'s work [23] was also based on standard GA, but they used toy problems of very small genome length of 100 base-pairs. The main difference between our work and other published works based on GA is in the definition of fragments. The base-pairs of the fragments used in Parsons' and others' works are fully read. They used GenFrag [24] to generate such fragments. In reality, by Sagner technique, we can read only small portions on both sides of the fragment. In our experiments (including our previous work [25]) a small portion of base-pairs, only at the two ends of the fragment, are known. Thus we handle a problem which is more realistic and difficult compared to the case where the base-pair sequence of the whole fragment is known. In addition, due to the use of such fragments, we could realize scaffolding in our proposed method.

Several deterministic algorithms, based on graph-theory, and greedy heuristic algorithms are proposed. But they are extremely computationally involved and need large scale parallel processing computational environment which is very costly. Worldwide only a few such installations are available, and they are owned by large research facilities. Yet, the need for genome sequencing is felt more and more strongly at every small medical research centers, drug development centers, agricultural research centers, and so forth. To help progress of their researches we need an efficient fragment assembling algorithm, which could run on an inexpensive computational platform. Moreover, on many occasions what one needs is only a partial sequencing, or to know whether a particular sequence is present in the genome or not, not the whole genome sequence.

The main motivation of this work is to find an efficient fragment assembling algorithm that could run on desktops, yet be able to find nearly correct draft sequences.

In the proposed method, fragment matching, contig formation, and scaffolding all are embedded in one process. Moreover if the researcher needs to know/confirm only certain gene sequence, information of which is available in the draft sequence (in the contig pool defined in Section 3.4), she/he may stop the moment it appears in the contig pool instead of continuing more generations of genetic search.

The rest of the paper is organized as follows. In Section 2, shotgun sequencing and problems of the existing techniques are briefly explained. Section 3 is devoted to explain the proposed algorithm. In Section 4, we state the experimental setup and results of the experiments using two actual genome sequences and discuss them. Conclusion is in Section 5.

2. Shotgun Sequencing Method

2.1. Shotgun Sequencing. In this section, we explain fragment assembling based on Sanger sequencing. To decode a long DNA sequence one needs to clone it to a few copies, split it up into fragments, read the individual fragments, and then assemble them in correct sequence to reconstruct the target DNA. This process is called shotgun sequencing and is the basis of all sequencing strategies. In 2000 Myers et al. successfully sequenced the fruit fly *drosophila* genome of length ≈125 Mbps using whole genome shotgun sequencing (WGSS) [16], and consequently WGSS was established as the generally accepted technique.

WGSS was used in the determination of draft human genome in 2001 by Celera Genomics [26]. In recent years, WGSS also decoded several biotic genomes, the chimpanzee by Mita in 2005 [27], the honeybee *Apis mellifera* by Weinstock in 2006 [28], and the Sea Urchin *Strongylocentrotus purpuratus* by Sodergren et al. in 2006 [29].

Our target is similar, to create the sequence based on WGSS, doing the assembling part using GA.

2.2. Outline of WGSS. The whole process of WGSS is divided into two steps—one is the biological part of cloning, fragmenting, and reading. The other one is the computational part of assembling the fragments.

2.2.1. Biological Part. The basic shotgun procedure starts with a number of copies of DNA whose sequence is cut into a large number of random fragments of different lengths. Fragments that are too large or too small are discarded. Of the remaining fragments, that is, those used for assembling, the length of short ones is about 2 kbp and of the long ones is about 10 kbp [11]. The base-pair at both ends of all the fragments is read with DNA sequencer, shown as dark parts

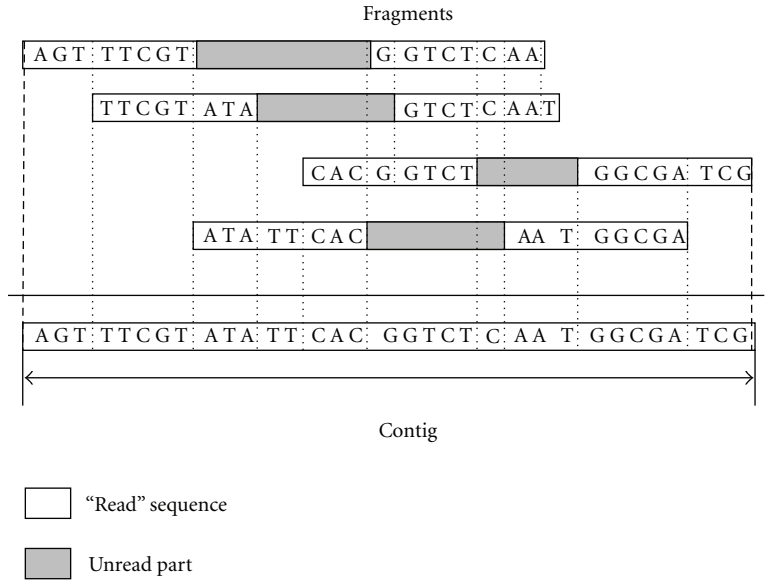

Figure 1: Formation of contigs.

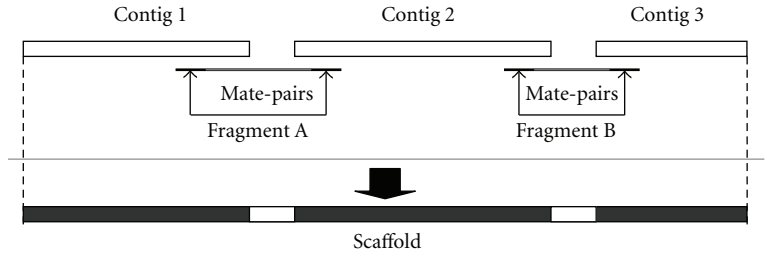

Figure 2: Scaffolding.

of fragments in Figure 1. Only about 500 to 1000 bp can be read using present sequencer technology. The base sequence at both ends of a fragment read by the sequencer is called *read*, and a pair of reads from two ends of a fragment is called *mate-pairs*.

Starting with a fair number of clones, the total base-pair reads of fragments are several times the number of bases in the original genome. Commonly, a term *coverage* is used to measure the redundancy of the fragment read data. It is defined as the total number of base reads from fragments as a ratio of the length of the source DNA [30]:

$$\text{Coverage} = \frac{\sum_{i=1}^{N} \text{reads\_of\_fragment}_i}{\text{target\_genome\_length}}, \quad (1)$$

where N is the total number of fragments. The genome is fragmented randomly. Their *read* parts together may not even include the whole genome if the *coverage* is low. To be able to reconstruct the original genome, the *coverage* needs to be set at around 8 to 10 (described as 8X∼10X). If coverage is high, the probability of covering original genome is higher and the accuracy of the assembled parts is improved. However, the number of fragments and therefore the computational complexity is also increased. Even though the *coverage* is 10X, some part of the original genome may not be available in the fragment *reads*.

2.2.2. Computational Part. To sequence the original DNA, we first identify overlapping sections by comparing the already read base sequences at both ends of the fragments, as shown in Figure 1. Long lengths of base sequences without gaps, obtained by assembling the *reads*, are called contigs. Figure 1 shows how two contigs are formed. Here, it is presumed that two overlapping *reads*, one a prefix of a fragment and the other the suffix, originate from the same region of the genome. This is however true only when the overlapped *base-pair* length is sufficiently long, as small sequences appear repetitively in the genome.

Two *reads* on two sides of a single fragment are called *mate-pair*. The position and distance between contigs are determined from the mate pair of fragments (Figure 2). Thus, subset of contigs with known order is grouped together and this process is called scaffolding. This is done by constructing a graph in which the nodes correspond to contigs, and a directed edge links two nodes when mate-pairs bridge the gap between them. Most of the recent assemblers include a separate scaffolding step. A rough frame of original genome sequence is made by this scaffolding process. After

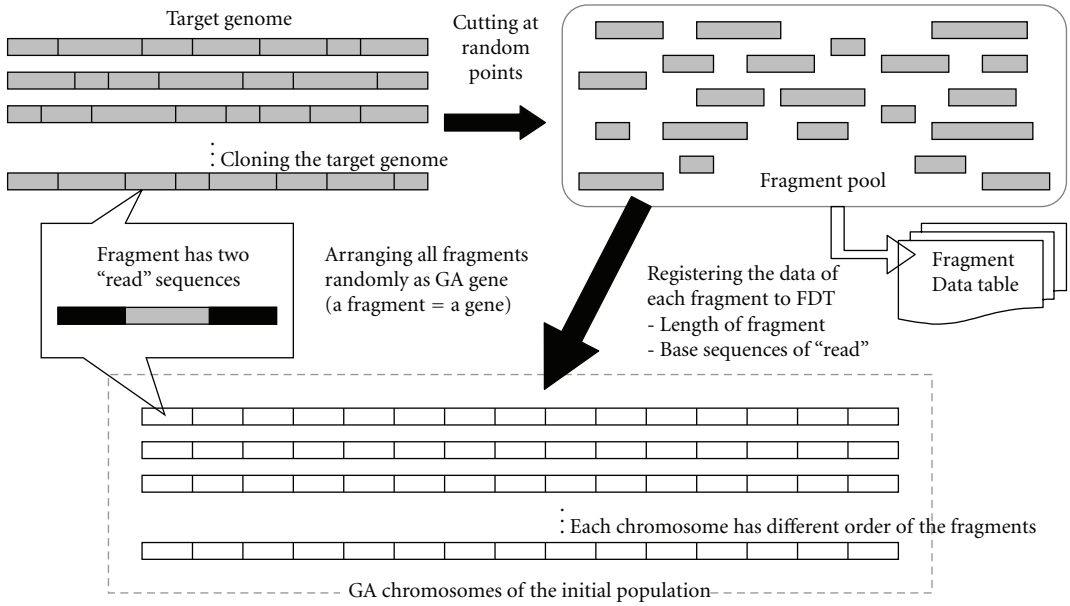

Figure 3: Chromosomes of GA for fragments assembly.

all contigs are oriented and ordered correctly, we can close gaps between two contigs. This process is called gap closer or finishing.

Celera assembler [26] employs scaffolding algorithm based on graph theory using mate pairs. TIGR assembler [14] employs a greedy heuristic algorithm.

3. Proposed Algorithm

Our proposed genetic algorithm technique is specialized for fragments assembling and similar problems. The main contribution here is Chromosome Reduction Step (CRed), which reduces the length of GA chromosome with progressive generation. As the chromosome length reduces, so does the search space, and the searching is more and more efficient. The other contribution is Chromosome Refinement Step (CRef), which is a greedy mutation to improve the correctness of the solution by local rearrangement of genes. We were able to combine the phase of overlap (contig formation) and scaffolding by the way we defined the structure of the GA chromosome and CRed. The details are explained in the following sections.

3.1. The Structure of GA Chromosome. Genes of our GA chromosome are genome fragments, where one fragment is one gene. In a GA chromosome gene, there is the information of two *read* sequences which are mate-pair, and the gap of unknown length in between. This fragment structure is different from previous works, where the *base-pair* array of the whole fragment is known. This makes our problem more realistic and complex.

The fragments generated by shotgun sequencing method are labeled in serial numbers, 1 to N, where, N is the total number of fragments. The read information corresponding to different fragments is stored in Fragment Data Table (FDT). The chromosome is composed of all these N fragments sequenced in random. Thus a chromosome is actually a permutation of numbers 1 to N, which are labels of different fragments. A number of such chromosomes, equal to the population size, are created. The formation of the initial population of GA chromosome is illustrated in Figure 3. A fragment is not cut on the way of genetic operations like crossover, because crossover and mutation are done at the boundary of fragments (genes of the chromosome). Thus, the information of mate-pair is retained without break. The flow of genetic search algorithm is shown in Figure 4. Different blocks of the algorithm are explained below in Section 3.2 to Section 3.5.

3.2. Evaluation Function. The goal of the search is to bring closer the fragments generated from the same region of the original chromosome. Fitness of a GA chromosome increases as adjacent genes match in their *base-pair* arrays. Similarities of all adjoining pairs of GA chromosome genes (which are actually the genome fragments) are calculated to find the fitness as follows:

$$\text{Fitness}(c) = \sum_{i=0}^{N-2} \text{similarity}\ (i, i+1). \tag{2}$$

The genes in a chromosome are numbered 0 to $N-1$, from left to right. In (2), i and $i+1$ are adjoining fragments and N is the total number of genes in the chromosome. To calculate the similarity (overlap), we use Smith-Waterman algorithm [31] that detects alignment of a pair of genes by dynamic programming. Here we set a threshold value mp of overlap to judge whether there is a real match between two fragments. If two fragments i and $i+1$ have the same sequence of bp

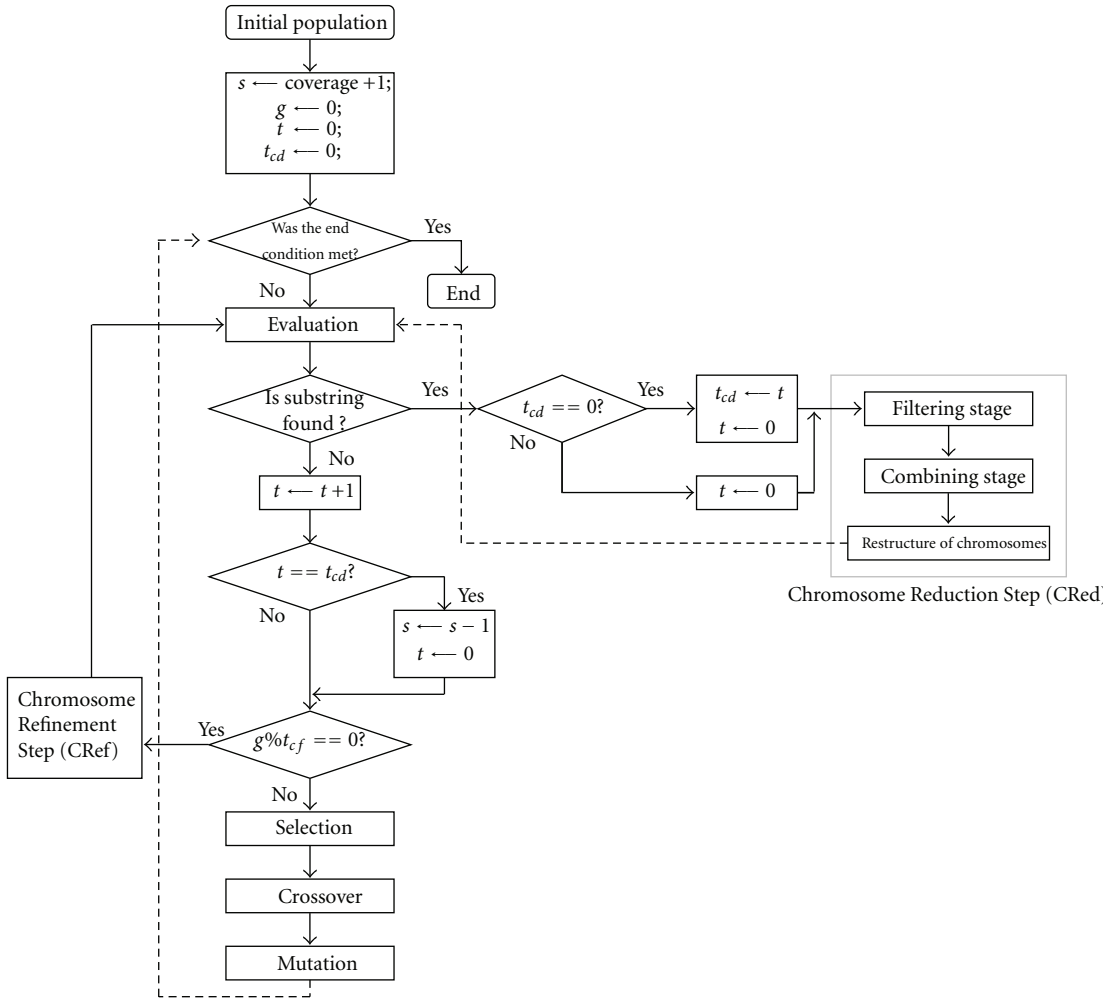

Figure 4: Algorithm including CRed and CRef.

of length more than or equal to mp, similarity $(i, i+1)$ score is 1, and otherwise it is 0. Similarity $(i, i+1)$ could have discrete values 0, 1, or 2. Because each fragment has two *reads*, similarity is 2 if two *reads* on both sides of the fragments matched. If the *reads* on only one side matches, the score is 1. If we set mp low, the misassembling (error) probability increases. On the other hand, if we set mp high, the probability of successful match would be low slowing the progress of genetic search.

3.3. Selection, Crossover, and Mutation

3.3.1. Selection. We use roulette-wheel selection and elitist preservation. Roulette-wheel selection tends to converge on local maximum when a few chromosomes have much better fitness. On the other hand, the selection is more fair as it properly takes care of individuals' fitness. Other selection methods, like ranking and tournament selection, have less selection pressure and therefore less probability to early convergence. The variance of fitness among chromosomes is low in our chromosome design, and due to CRed operation. (At the end of CRed operation every chromosomes fitness again resets to a low narrow range). In fact, in a preparatory experiment we have verified that roulette-selection is more efficient for the proposed algorithm.

3.3.2. Crossover and Mutation. We do not allow multiple copies of the same fragment in our GA chromosome. To ensure that, we used order-based crossover (OX) and swap, often used in solving TSP [32]. In OX, *offspring* 1 directly copies genes from *parent* 1, from the beginning up to the crossover point. The rest of the genes are copied from *parent* 2 preserving the sequential order of *parent* 2 and skipping the genes already copied from *parent* 1. *Offspring* 2 is constructed similarly. Here, two point crossover is also possible, but we used one point crossover.

Mutation is done by simple swapping. Two genes in a chromosome are selected at random and swapped over. In swap mutation, it is also possible to swap a subset where the selected gene is included in the subset. By doing so we can avoid breaking the subset already formed. But we did simple one pair gene swapping.

3.4. Chromosome Reduction (CRed) Step. Through generations, chromosomes bring individual fragments with long matched *base-pairs* to adjacent positions by evaluation function and selection. Once overlapping fragments are brought closer, we use CRed operation to separate out formed contigs and reorganize array of genes in the chromosome. This is done in two stages, filtering stage and combining stage. These two stages together is called Chromosome Reduction Step (CRed).

In filtering stage we search for contigs already formed in GA chromosome. The search is performed on the elite chromosome. If contig over a certain threshold length is formed, all fragments contained within that contig are extracted from all chromosomes. This shortens the length of chromosomes.

Further detail is as follows. Here, s is the threshold length of the target substring, expressed as number of fragments. g is the generation number. t is a counter for counting generations. The threshold length s is reduced from its initial value, as formation of longer substring become more and more difficult as GA generation progresses. t_{cd} is the interval in numbers of generations, which determines when s should be decreased.

First, s is initialized with the value (coverage + 1). g and t are also initialized at 0 and increments with each generation. We search for the subset/s consisting of s fragments in the best chromosome, after fitness evaluation. If such subset is not found, CRed is not started and $g \leftarrow g + 1$ and $t \leftarrow t + 1$. First time such subset/s is/are found, mark the fragments which are contained within the subset/s. Those fragments are deleted from all the chromosomes, and Fragment Data Table (FDT) is updated. At the same time, $t_{cd} \leftarrow t$ and $t \leftarrow 0$.

Marked fragments are combined based on their overlaps. The contig/s is/are stored in a separate database that we call "contig pool" which is indexed in Contig Data Table (CDT). This stage is called combining stage. If the other contig/s is/are already in the contig pool, newly formed contig is compared with those contig/s and is combined with those to get longer contigs whenever possible. Accordingly "contig pool" and CDT are updated.

As mentioned, the length of subset to be extracted, in terms of number of fragments, is initialized to $s =$ (coverage + 1). At every generation all subsets of length (coverage + 1) fragments are extracted from the best chromosome. If no such subset is assembled for consecutive t_{cd} generations, since when a contig consisting of s fragments was found, s is reduced by 1. The flow of the CRed algorithm is shown as "yes" part of the decision diamond "*is substring formed?*".

The first time, when s is initially set to (coverage + 1) and is the longest, number of generations taken for contig formation is also longest. We set t_{cd} as number of generations we allow for new substring of length s to be formed. As long as we find contigs of length s within t_{cd} generations, s is not reduced. If, even after t_{cd} generations, subset of length s is not formed, we reduce our expectations to subset of smaller lengths. We decrement the parameter s by one, as well as $t \leftarrow 0$. Once a substring is found, the filtering is done by which fragments corresponding to the newly assembled substring are deleted from all chromosomes.

After filtering stage, combining stage is executed. When a new contig is added to the contig pool, we try to combine it with the existing contigs, if possible, to make longer contigs. Once a longer contig is formed, further genes (genome fragments) could be shed off from the chromosomes the way it is done in the filtering stage.

In filtering stage of CRed, the fragments in the substring extracted from the chromosome, may join to one end of an existing contig, or it may join two contigs on two sides to form a very long contig. Information of contigs after filtering operation is updated to Contig Data Table (CDT). Information about their relationship, if any, obtained from *mate-pairs* are added to the Scaffold Data Table (SDT). Thus, SDT holds the information about the label of contigs and their relative positions. Every time combining stage starts, new contigs are compared with existing contigs and combined when possible. CDT and SDT are renewed after that. Using simple user interface, the formation of contigs and scaffolds can be visualized and it is possible to manipulate them manually by the user or an expert, when available.

As the contigs become longer and chromosomes shorter, GA runs more efficiently. After every combining stage, the user could check whether the available results are good enough (long enough) for her/his purpose. If not, the genetic search continues.

3.5. Chromosome Refinement (CRef) Step. Instead of depending on genetic search alone, we add a step to facilitate proper sequencing more efficiently by manual greedy swapping. This is a simple and fast heuristic that we named CRef.

CRef improves the quality of solution by rearranging the sequence of fragments in a GA chromosome to correspond to the base sequence in the target genome. When two fragments A and B are sequentially positioned in a chromosome due to high-degree of overlap, the following overlap patterns, as shown in Figure 5, are possible.

If two fragments have overlap of pattern 4, it is obvious that their sequential order is wrong in the chromosome. We swap the positions of these two fragments. With this, the positions of fragments in GA chromosome are arranged to correspond to their positions in the original genome as shown in Figure 6.

This concept could be extended by expanding the scope of fragment comparison, beyond that of adjacent fragments only. We set a numeric parameter f_{cf} which represents the scope of comparison of fragments. For example, when $f_{cf} = 3$, all three neighboring fragments are compared. Though, while evaluating the fitness of a GA chromosome, we compare two adjoining fragments only, it is possible to find higher similarity with a fragment which is one or two fragments away. This could be explored by setting f_{cf} to a value more than 2 and running CRef.

A detail explanation of the CRef operation is as follows. Here, g is the number of generations, and t_{cf} is the parameter specifying the interval, in number of generations,

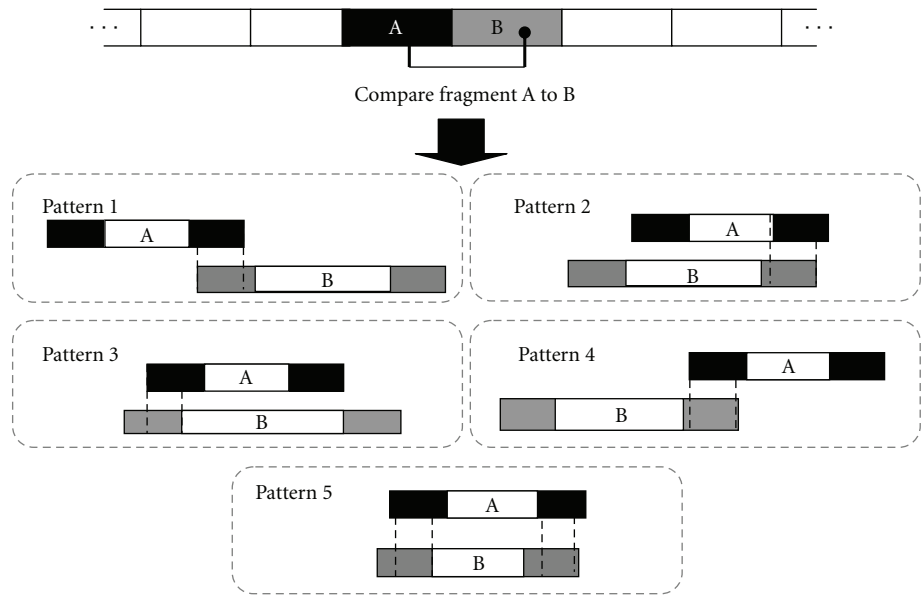

Figure 5: Matching pattern of two fragments. Pattern 1: tail-part of fragment A overlaps with the beginning of fragment B. Pattern 2: tail-parts of the two fragments overlap. Pattern 3: beginning of the two fragments overlap. Pattern 4: beginning of fragment A overlaps with end of fragment B. Pattern 5: both beginning and end of the two fragments overlap.

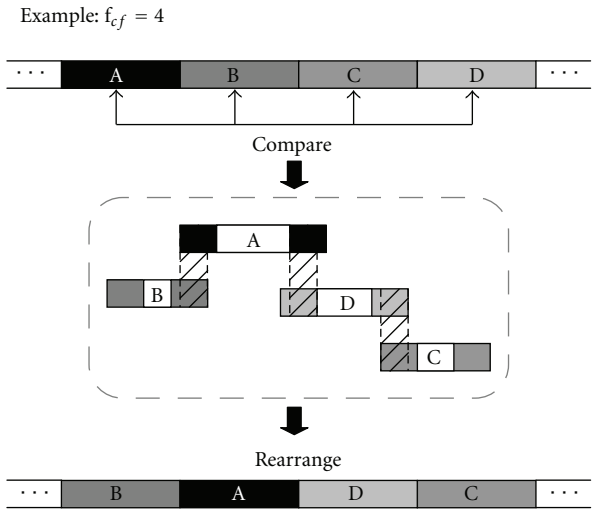

Figure 6: Chromosome Refinement (CRef) Step.

of the periodicity of execution of CRef. Thus, if t_{cf} is 100, CRef will be executed every 100 generations. N is the total number of gene in the chromosome. As the CRef operation is computationally heavy, it is run over the best few chromosomes in the population. We set a parameter rc which specifies the number of chromosomes on which CRef operation will be undertaken.

First, we set the values of f_{cf}, t_{cf}, and rc. Typical values are $f_{cf} = 3$, $t_{cf} = 100$, and $rc = 10$. At intervals of t_{cf} generations, we compare matching portions from fragment 1 to N, taking consecutive f_{cf} fragments at a time, from best rc chromosomes. If pattern 4 type matching (Figure 5) or fragments of high similarity (but not adjacent) are found, we rearrange them properly as shown in Figure 6.

The computation cost is low much because CRef is neither executed on all chromosomes nor is executed at every generation. It is limited to a few high-fitness chromosomes (set by the parameter rc) and the periodicity of execution is set by t_{cf}. These two parameters are set depending on the available computational power and time.

With these two steps of CRed and CRef, both the efficiency and quality of result of our genetic search greatly improved.

4. Experiments and Results

In this section, we describe the details of our experimental setup, discuss the results, and compare them with a the most frequently referred GA-based assembling proposed by Parsons et al. [20].

4.1. Experimental Genome Data. We used two real genome sequence data, also frequently used by other researchers, to test the effectiveness of our algorithm. They are available in the NCBI database [33]. The important features are shown in Table 1. POBF is the human apolopoprotein, which is 10089 bp long. AMCG is the initial 40% of the bases from LAMCG which is the complete genome of bacteriophage lambda and its length is 20100 bp. We scaled down number of fragments, fragment length, and *read* size compared to actual shotgun sequencing experiments to reduce the computation time. In actual experiment of shotgun sequencing, the genome fragments are of length 2 kbp~10 kbp and *read* lengths are 800 bp~1000 bp on both sides of the fragment. We cloned each genome sequence and splitted them into

Table 1: Experimental genome data.

POBF	AMCG
10089 bp	20100 bp
Accession no.: M15421	Accession no.: J02459
Human apolipoprotein B-100 mRNA, complete cds.	Bacteriophage lambda, complete genome (initial 40%)
Number of fragments: about 500	Number of fragments: about 1000

Table 2: The differences between proposed GA and Parsons' GA.

	Our GA	Parsons' GA
Gene of GA chromosome	Fragment with 2 *read*s	*read*s only
Fitness function	Equation (2)	
Crossover	Order-based crossover	
Mutation	swap mutation + greedy mutation	Swap mutation
Heuristic part	CRed and CRef	None
Scaffolding	possible	Not possible

fragments of length 200 bp to 500 bp imitating the shotgun method, but scaling down the length of a fragment by a factor of 10. Each *read* is set to 50 bp, scaled down by a factor of 12 to 20. Thus both the *read* and the fragment length are scaled down by similar factor compared to actual shotgun fragment assembly [11]. We also reduced the coverage from the standard value of 8X to 10X, so that the number of fragments is less. This reduces the computation load, but at the same time reducing the success rate of assembly the whole genome. There is further explanation in Section 4.2.

4.2. Experimental Setup. We implemented Parsons' GA-based algorithm and compared results with proposed algorithm under same experimental conditions. The basic differences between our GA and Parsons' GA are shown in Table 2.

In the experiment described in Parsons' paper, they used the some POBF and AMCG data. But the fragment lengths were different, and the whole fragment was readable. In our case, the *read* is only of small length (at the two ends) of the fragment—which is more akin to actual whole genome shotgun sequencing method and obviously more difficult. We ran both Parsons' and our algorithm on the same genome data. To reduce computation load, we set the *coverage* to 4X and 5X for POBF and AMCG, respectively, a much lower value compared to 8X~10X used in actual genome sequencing.

4.3. Setup of GA Parameters. Population size is set at 100 chromosomes which are generated by technique explained in Section 3.1. The crossover rate and the mutation rate are set at 0.8 and 0.05, respectively. In the experiment with POBF sequence, we ran genetic operation for 40 hours (about 2,000,000~2,500,000 generations), and 100 hours (about 5,000,000~6,000,000 generations) for AMCG experiment, using a desktop PC (CPU is Intel Xeon 3.40 GHz and 3.00 GB RAM). The threshold parameter mp (Section 3.2) is set at 25 bp. mp length of 50% of *read* is a strict setting. But a shorter mp could lead to error in the final assembled array from unwarranted matching. In fact, in other experiments, a much lower value of 5% (of the read which is 40~50) is generally used [26]. But if we set mp around 40 to 50, we could hardly get any match, with a *read* of length 50.

We defined s in Section 3.4. This parameter, which triggers the starting of CRed operation, is initially set at a value equal to (coverage + 1). The value of s is decreased in steps of 1 till $s = 2$.

In another preparatory experiment we examined how to set the proper value of the parameter f_{cf}, used in CRef. We experimented setting the value of f_{cf} between 2 to 5. Repeated experiments showed that $f_{cf} = 3$ gives the best result.

4.4. Results. We experimented 20 trials with different sets of fragments, with POBF and AMCG genome data. In our proposed technique, the number and length of contigs were checked in the "contig pool" (Section 3.4). The reconstruction ratio for the proposed algorithms as well as Parsons' algorithm is the percentage of base sequences which could be reconstructed in a certain period of time. We counted the number of bases of all obtained contigs. It is good if the reconstruction ratio is high. However even if the reconstruction ratio is high, a result with many gaps is not good. The number of contigs is that measure. Less number of contigs are better. If gaps are frequent, the results may be useless. Error is the number of the mismatches. Mismatch means fragments are combined incorrectly, that is, not the way they are in the original genome. This may happen when mp is chosen to be too short. The contigs generated using our algorithm and Parsons' GA, for POBF and AMCG data, are compared in Table 3. The value in parenthesis is the number of times 100% reconstruction is achieved.

With our proposed GA, we could reconstruct the complete original genome of POBF twice. Though Parsons' GA could obtain the complete genome in their paper [20], the experimental genome data used by them did not have any gap and the whole fragment was *readable*, making the problem less complex. With an unread gap in the fragment between two *reads*, the fragment data used in our experiment is realistic though reconstruction of the original genome sequence was more difficult.

Because fragmentation is done randomly, though the overall coverage was 4X, some part of the genome may be covered only once by the fragment *reads*. In Parsons' algorithm, if those fragments could not be matched to a combinable contig in the early stage, they are left alone by the other subsets already formed. This is because the fragment matching is tried for the adjoining fragments only. Those fragments are left alone, and only mutation can combine them. In contrast, our proposed technique has higher chance of combining those fragments. One reason is that, because we have two *reads* separated by an *unread* portion, it is unlikely that base-pairs in both *reads* are sparse in number. Moreover,

Table 3: Results about contigs.

	POBF				AMCG			
	Proposed GA		Parsons' GA		Proposed GA		Parsons' GA	
	Average	Best	Average	Best	Average	Best	Average	Best
Number of contig	27.6	1	19.8	9	38.3	8	35.1	11
Length of contig	317.3	10089	342.6	1008	349.4	2795	316.4	1341
Reconstruction ratio	86.8	100 (2)	67.2	74.3	66.5	72.1	55.2	61.1
Error	0	0	0	0	0	0	0	0

Table 4: Results about scaffolds.

	POBF		AMCG	
	Average	Best	Average	Best
Number of scaffold	4.1	1	12.4	6
Length of scaffold	2135.9	10089	1079.1	3888

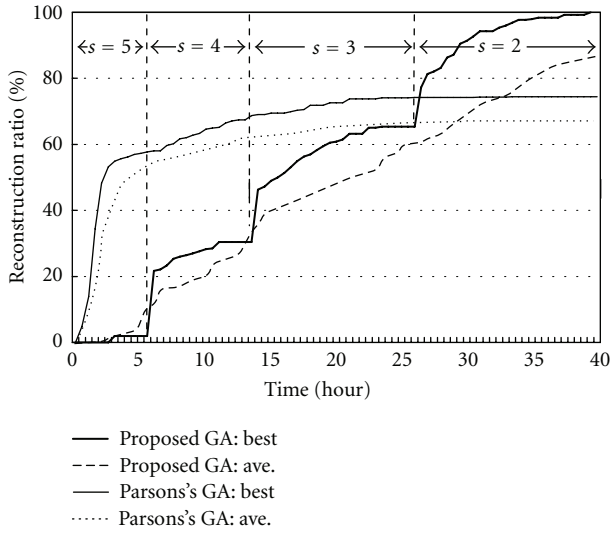

Figure 7: The improvement of the reconstruction ratio: POBF dataset.

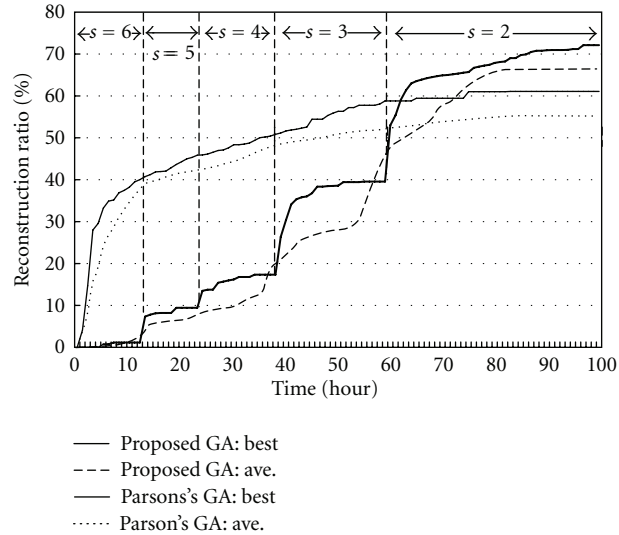

Figure 8: The improvement of the reconstruction ratio: AMCG dataset.

due to CRed operation, there are more realignments of fragments as well as shortening of the whole chromosome. They together improve the chance of combining the lone *reads*.

The average length of contigs using POBF data in our proposed GA is slightly lower than that of Parsons' GA. However, the scaffolds generated by our proposed technique, as shown in Table 4, are longer. The length of scaffold shown is the length of known contig parts only. Scaffolding is achieved as part of the proposed algorithm, which is a necessary step in shotgun sequencing.

The improvement of the reconstruction ratio versus execution time is shown in Figures 7 and 8 for POBF and AMCG, respectively. We checked the reconstruction ratio every 30 minutes for POBF and every hour for AMCG.

We calculated the reconstruction ratio from the total length of all contigs in the contig pool. It was 0% in the beginning until CRed started its operation. Because we set a threshold value for substring length to be taken out from chromosome and transferred to contig pool, improvement of reconstruction ratio was stagnant periodically. When the stagnation continued over a certain period of time, CRed parameter s is lowered to break the stagnation. At every such period, s is set high with the hope of getting long contigs. When it appears that such long contigs could not be constructed, s is reduced by 1. Then new contigs were found in the GA chromosome and the reconstruction ratio increased again. We can clearly observe such surges in the curve of reconstruction ratio. For POBF, there are four steps. Initially s was set to 5 (coverage + 1), and subsequently reduced to 4, 3, and 2. For AMCG, there are five steps, starting with $s = 6$ and finally reduced to $s = 2$.

Even at the end of the predefined length of execution time (40 hours and 100 hours), the results were improving. By increasing f_{cd} and the total length of execution time, better results could be obtained. Our algorithm would be able to get much better results with actual fragment assembly data of high coverage, longer, *reads* and lower value of mp (in percentage of *read*). We could not perform such experiments due to lack of computation resource.

5. Conclusion

We proposed a genetic-algorithm-based approach to assemble DNA fragments to construct the genome sequence. Our GA chromosomes were different from previous approaches. We also added two modifications, Chromosome Reduction step (CRed) and Chromosome Refinement Step (CRef), to improve the efficiency of GA optimization for fragment assembly. Experimenting with actual genome data, we could obtain 100% of the POBF genome sequences. We compared our proposed algorithm with Parsons's algorithm and have shown that the proposed algorithm delivered better results.

We used a coverage of only 4X instead of more practical ≈10X. More fragments covering the same part of the sequence are required for better reconstruction ratio and accuracy. Also, introducing biological knowledge is helpful for effective fragment assembling and improving its accuracy, which, in future, we will link with our genetic search, in CRed and CRef operations. In amino acid and protein formation, there are some distinct rules in the alignment of bases. Using that pattern knowledge we could decrease the computational cost during similarity measurements in addition to dynamic programming. We plan to include domain knowledge to design more efficient dynamic programming specific to this problem. In fact, if we know the threshold length *mp*, it is much efficient to compare blocks of nucleotides of length *mp*. There is another drawback in our approach. In chromosome refinement stage, we evaluate the degree of matching using the way the two fragments are oriented in the GA chromosome. In fact, the orientation may be reversed, from the way they appear in the actual genome. Comparing both orientations for neighboring fragments would be computationally more complex, but we can achieve the whole genome structure in less number of GA generations, as well as with less number of fragments. These two modifications are our future work.

Though CRed step is proposed for fragment assembling problem, it is applicable to similar problems like clustering, path search and other combinatorial optimization.

References

[1] J. C. Neil and P. A. Pavel, *An Introduction to Bioinformatics Algorithms*, A Bradford Book, 2004.

[2] Y. Li and J. C. Patra, "Genome-wide inferring gene-phenotype relationship by walking on the heterogeneous network," *Bioinformatics*, vol. 26, no. 9, Article ID btq108, pp. 1219–1224, 2010.

[3] Y. Li and J. C. Patra, "Integration of multiple data sources to prioritize candidate genes using discounted rating system," *BMC Bioinformatics*, vol. 11, no. 1, article S20, 2010.

[4] E. S. Lander, L. M. Linton, B. Birren et al., "Initial sequencing and analysis of the human genome," *Nature*, vol. 409, no. 6822, pp. 860–921, 2001.

[5] T. Sasaki, "The map-based sequence of the rice genome," *Nature*, vol. 436, no. 7052, pp. 793–800, 2005.

[6] J. D. Watson and F. H. C. Crick, "Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid," *Nature*, vol. 171, no. 4356, pp. 737–738, 1953.

[7] P. A. Benjamin, *Genetics—A Conceptual Approach*, W. H. Freeman and Company, 2005.

[8] P. P. Vaidyanathan, "Genomics and proteomics: a signal processor's tour," *IEEE Circuits and Systems Magazine*, vol. 4, no. 4, pp. 6–29, 2004.

[9] W. C. Warren, L. W. Hillier, J. A. Marshall Graves et al., "Genome analysis of the platypus reveals unique signatures of evolution," *Nature*, vol. 453, no. 7192, pp. 175–183, 2008.

[10] F. Sanger, A. R. Coulson, G. F. Hong, D. Hill, and G. Petersen,, "Nucleotide sequence of bacteriophage λ DNA," *Journal of Molecular Biology*, vol. 162, no. 4, pp. 729–773, 1982.

[11] M. Pop, "Shotgun sequence assembly," *Advances in Computers*, vol. 60, pp. 193–248, 2004.

[12] M. T. Tammi, *The Principles of Shotgun Sequencing and Automates Fragment Assembly*, Center for Genomics and Bioinformatics, Karolinska Institute, Stockholm, Sweden, 2003.

[13] S. Kim, "A survey of computational techniques for genome sequencing," Project Report, Korea Institute of Science and Technology Information, 2002.

[14] G. G. Sutton, O. White, M. D. Adams, and A. R. Kerlavage, "TIGR assembler: a new tool for assembling large shotgun sequencing projects," *Genome Science and Technology*, vol. 1, pp. 9–19, 1995.

[15] K. Mita, M. Kasahara, S. Sasaki et al., "The genome sequence of silkworm, Bombyx mori," *DNA Research*, vol. 11, no. 1, pp. 27–35, 2004.

[16] E. W. Myers, G. G. Sutton, A. L. Delcher et al., "A whole-genome assembly of *Drosophila*," *Science*, vol. 287, no. 5461, pp. 2196–2204, 2000.

[17] X. Huang and A. Madan, "CAP3: A DNA sequence assembly program," *Genome Research*, vol. 9, no. 9, pp. 868–877, 1999.

[18] P. Green, "Phrap documentation : algorithms," Phred/Phrap/Consed System Home Page, April 2006, http://www.phrap.org/.

[19] L. Li and S. Khuri, "A comparison of DNA fragment assembly algorithms," in *Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Sciences (METMBS '04)*, pp. 329–335, June 2004.

[20] R. J. Parsons, S. Forrest, and C. Burks, "Genetic algorithms, operators, and DNA fragment assembly," *Machine Learning*, vol. 21, no. 1-2, pp. 11–33, 1995.

[21] K. Kim and C. K. Mohan, "Parallel hierarchical adaptive genetic algorithm for fragment assembly," *IEEE Congress on Evolutionary Computation*, vol. 1, pp. 600–607, 2003.

[22] E. Alba, G. Luque, and S. Khuri, "Assembling DNA fragments with parallel algorithms," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC '05)*, vol. 1, pp. 57–64, September 2005.

[23] S. C. Fang, Y. Wang, and J. Zhong, "A genetic algorithm approach to solving DNA fragment assembly problem," *Journal of Computational and Theoretical Nanoscience*, vol. 2, no. 4, pp. 499–505, 2005.

[24] M. L. Engle and C. Burks, "Artificially generated data sets for testing DNA sequence assembly algorithms," *Genomics*, vol. 16, no. 1, pp. 286–288, 1993.

[25] S. Kikuchi and G. Chakraborty, "Efficient assembling of genome fragments using genetic algorithm enhanced by heuristic search," in *Proceedings of IEEE Congress on Evolutionary Computation (CEC '07)*, vol. 1, pp. 305–312, September 2007.

[26] J. Craig Venter, M. D. Adams, E. W. Myers et al., "The sequence of the human genome," *Science*, vol. 291, no. 5507, pp. 1304–1351, 2001.

[27] K. Mita, "Initial sequence of the chimpanzee genome and comparison with the human genome," *Nature*, vol. 437, no. 7055, pp. 69–87, 2005.

[28] G. M. Weinstock, G. E. Robinson, R. A. Gibbs et al., "Insights into social insects from the genome of the honeybee *Apis mellifera*," *Nature*, vol. 443, no. 7114, pp. 931–949, 2006.

[29] E. Sodergren, G. M. Weinstock, E. H. Davidson et al., "The genome of the sea urchin Strongylocentrotus purpuratus," *Science*, vol. 314, no. 5801, pp. 941–952, 2006.

[30] J. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology*, PWS Publishing Company, 1997.

[31] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of Molecular Biology*, vol. 147, no. 1, pp. 195–197, 1981.

[32] Z. Michalewicz, *Algorithms + Data Structures = Evolution Programs*, Springer, 1999.

[33] The National Center for Biotechnology Information, Home page at http://www.phrap.org/.