An Enhanced Genetic Algorithm for DNA Sequencing by Hybridization with Positive and Negative Errors

Thang N. Bui and Waleed A. Youssef

Department of Computer Science
The Pennsylvania State University at Harrisburg
Middletown, PA 17057
{tbui,wuy101}@psu.edu

Abstract. This paper describes a genetic algorithm for the DNA sequencing problem. The algorithm allows the input spectrum to contain both positive and negative errors as could be expected from a hybridization experiment. The main features of the algorithm include a preprocessing step that reduces the size of the input spectrum and an efficient local optimization. In experimental tests, the algorithm performed very well against existing algorithms. The algorithm also performed very well on a large data set generated in this paper from real genomes data.

1 Introduction

Determining the genome of living organisms has been a major research initiative world wide in the last few years. One of the principal steps in this endeavor is the sequencing of DNA. Informally, DNA sequencing is the process of determining the correct order of nucleotides in a DNA segment. Many techniques have been developed for DNA sequencing. DNA sequencing experiments are typically performed in two stages: shotgun sequencing and walking. In shotgun sequencing, many short, randomly selected fragments of a DNA segment are sequenced. Due to the stochastic nature of this process, there are parts of the DNA segment that are left unsequenced or insufficiently covered. These parts are then covered by a deterministic finishing process called walking [19].

The two most popular methods for DNA sequencing are the Sanger method and the Sequencing by Hybridization (SBH) method [20]. In this paper we consider only the SBH method. SBH follows the methodology known as "break, read and assemble". In this methodology a DNA sequence is partitioned into smaller size fragments. The fragments are then read using a fluorescent light. The assemble phase tries to retrieve the original sequence from the shorter length fragments, i.e., to determine the exact sequence of nucleotides of the DNA molecule.

From an algorithmic point of view the DNA sequencing problem is the problem of constructing a chromosome that most likely contain all the DNA fragments in a given input set, called the *spectrum*. The spectrum is usually obtained through some experiments such as a hybridization experiment. It should be noted that the fragments in a spectrum have overlaps and all fragments have the same length. If a spectrum contains all possible fragments of length l of a DNA sequence and there are no errors in the fragments then there exist efficient algorithms for reconstructing the original DNA sequence from the spectrum. In general, however, there are errors in the spectrum, e.g., missing fragments or erroneous fragments, making the problem of reconstructing the original DNA sequence an NP-hard problem [4].

In this paper we present a genetic algorithm for the DNA sequencing problem. Our algorithm differs from others in that it can efficiently handle different types of errors in the input. Additionally, our algorithm includes a preprocessing step that effectively reduces the size of the input, thereby helps reduce the running time of the algorithm. The idea of the preprocessing step can be extended to create a hierarchical structure that enables our algorithm to deal with much longer sequence. Experimental results show that our algorithm outperformed other algorithms from [1] and [7]. We also performed extensive test of our algorithm on data that we generated systematically from genomes obtained from the GenBank (www.ncbi.nlm.nih.gov/Genbank/). To determine the quality of these results we used the Smith-Waterman algorithm for sequence alignment [24]. The results of these experiments show that our algorithm is very robust against a large range of errors.

The rest of the paper is organized as follows. Section 2 describes some common terminologies, defines the problem formally and lists some current work on the problem. The algorithm is described in Section 3. Experimental results comparing the performance of our algorithm against others are given in Section 4. Section 4 also includes results showing the performance of our algorithm on a large data set that we generated. The conclusion and future directions are given in Section 5.

2 Preliminaries

In this section we describe some of the terminologies needed for the rest of the paper, give a formal description of the DNA sequencing problem and list some of the existing works that have been done on this problem. DNA (deoxyribonucleic acid) consists of two strands, each of which contains *nucleotides*: adenine (A), cytosine (C), guanine (G) and thymine (T). (Technically, there are other components in a DNA strand such as phosphates.) The nucleotides in each strand are connected together in series. The two strands of the DNA are twisted together into the famous double helix structure. Furthermore, each nucleotide in a strand is connected to a *complementary* nucleotide in the other strand, where A is paired with T and C is paired with G. Thus, each strand in a DNA completely determines the other.

A *fragment* is a short sequence of nucleotides. It is also known as an *oligonucleotide*. A *hybridization experiment* is an experiment that takes a DNA strand and produces copy of fragments of that strand. These fragments usually have overlap. The set of all fragments that result from a hybridization experiment is known as a *spectrum.* All fragments in a spectrum have the same length. In this paper, we consider fragment lengths ranging from 10 to 50.

The DNA sequencing problem is the problem of determining a DNA strand based on a given spectrum. We can model this problem as follows. Let $\Sigma = \{A, C, T, G\}$ be an alphabet. Here we consider a spectrum as a set of distinct strings of length l over Σ. We refer to each string in a spectrum as a *fragment.* A spectrum is said to be *ideal* if the following condition is true for all but one fragment in the spectrum: the suffix of length $l - 1$ in a fragment is a prefix of exactly one other fragment in the spectrum. The DNA sequencing problem can then be stated as the problem of constructing a string over Σ from a given spectrum (not necessarily an ideal spectrum), so that the resulting string is the shortest string that contains as many of the fragments in the spectrum as possible.

In general the input spectrum is not an ideal one. The errors appearing in a spectrum are usually due to errors in the hybridization experiment. Errors can be classified as positive or negative. The spectrum has *positive errors* when it contains fragments that are not part of the original sequence. It has *negative errors* when it fails to contain some oligonucleotides. Certain errors are *random*, meaning that they may disappear when the experiment is repeated. However, many hybridization errors are *systematic*, meaning that they are likely to repeat each time the experiment is run [22][23].

If there are no errors, the problem of DNA sequencing is similar to the Shortest Superstring problem [20], which is defined as the problem of reconstructing a string given a collection of overlapped substrings. The Shortest Superstring problem is NP-hard, but it is known that greedy algorithms work well for this problem [12]. There exists an approximation algorithm with an approximation factor of three, i.e., the superstring it produces is at most three times as long as the optimal shortest superstring [8]. However, compared to the DNA Sequencing problem, the Shortest Superstring problem seems to be easier.

The existence of errors in the input spectrum makes the problem of reconstructing the original sequence an NP-hard problem [4]. Missing fragments from the experiment turn the problem into the problem of finding the Most-Likely sequence [4]. The most likely sequence is the shortest one containing almost each fragment as a substring. Some fragments might be excluded from the final result. Those excluded fragments are the ones that represent the positive errors in the experiment. Also, fragments might not be completely overlapped. Under normal situations, two fragments of length l intersect in $l - 1$ positions. However, because of negative errors the longest overlap might not be of length $l-1$. Another source of difficulty exists when the spectrum contains repeated fragments. Most existing algorithms that allow for errors in the input spectrum put restrictions on the error model [5][10][11]. There are few algorithms that do not have any restriction on the input error model. Two such algorithms are in [1] and [7], and our algorithm is compared against them.

Algorithms solving the DNA Sequencing problem take as input the spectrum of all fragments. The output is a DNA sequence that is the most-likely one that includes all fragments. In the case of an ideal spectrum, the result sequence would be of length $n + l - 1$, where n is the number of fragments in the spectrum, l is the length of the fragments. However, because of errors this may not be always the case. The algorithm presented here deals with errors. So the output sequence would not necessarily be of length $n+l-1$. Negative errors may cause the output sequence to be shorter in length. Positive errors may cause it to be longer. In some other cases, those types of errors would mistakenly cause the algorithm to converge to a sequence that does not necessarily represent the optimal solution.

3 Algorithm

In this section we describe a genetic algorithm for solving the DNA sequencing problem when the input may have both positive and negative errors. We do not require that the starting fragment of the sequence be known as it is done in [6]. We use a steady state genetic algorithm together with a local optimization procedure to help improve the performance of the algorithm. Additionally, we have a preprocessing step that helps improve the algorithm even further. The overall algorithm is given in Figure 1. In the following subsections we give more details of the algorithm.

```
Sequence($S$)   // $S$ is a spectrum
  preprocess($S$)
  generate a random initial population $P$
  for each $a \in P$
     LocalOptimize($a$)
  endfor
  repeat
     Select two parents $p_1$ and $p_2$
     $u$ $\longleftarrow$ crossover($p_1$, $p_2$)
     LocalOptimize($u$)
     mutate($u$)
     replace($u$, $p_1$, $p_2$, $P$)
  until (there is no improvement)
  return the best member of $P$
  Align output sequence;
```

Fig. 1. The Enhanced Genetic Algorithm for DNA sequencing

Preprocessing. In general, the fewer fragments and longer fragments there are in the spectrum the easier the problem is. The idea of the preprocessing is to merge certain fragments together, thereby creating a new spectrum that has fewer and longer fragments. In this step we create long chains of fragments of the form $F_1 \dots F_k$, where F_i's are fragments, and the last $l-1$ elements of F_i match the first $l-1$ elements of F_{i+1}. Our objective is to make k as large as possible. The optimal case would be when $k = n$, where n is the cardinality of spectrum S. However, because of negative errors, that's not always the case. Each such chain of fragments is merged into one fragment in the new spectrum. The algorithm then works with this spectrum which has variable length fragments and a smaller number of fragments than the original spectrum.

The preprocessing algorithm creates a chain by selecting an unused fragment in the spectrum and adding it to the chain. The fragment is then marked used. The algorithm extends the chain by selecting an unused fragment that has an overlap of $l-1$ with the last fragment in the chain. If such a fragment exists, it is added to the chain and marked used, and the process is repeated. If there is no such fragment the chain is terminated. The algorithm then starts a new chain. The algorithm terminates when all fragments in the original spectrum have been used. The algorithm then merges the fragments in each chain to create a fragment for the new spectrum. This algorithm can be efficiently implemented using dynamic programming technique.

As an example, suppose we have a DNA sequence CTAGACGTTC of length 10. An ideal spectrum would consist of the following six fragments: CTAGA, TAGAC, AGACG, GACGT, ACGTT and CGTTC, where we have assumed that the fragment length is 5. However, because of errors from the hybridization experiment an input spectrum in this case may consists of the following six fragments: CTAGA, TAGAC, AGACG, TATCC, ACGTT, CGTTC. This spectrum differs from the ideal spectrum in that it does not contain the fragment GACGT (a negative error), instead it contains the fragment TATCC which is not a substring of the original DNA sequence (a positive error). Thus, this spectrum has one negative error and one positive error. Using this spectrum as the input, the preprocessing algorithm would produce the following chains [CTAGA, TAGAC, AGAC], [TATCC], and [ACGTT, CGTTC], which yields the spectrum consisting of the following three fragments: CTAGACG, TATCC, ACGTTC.

Encoding and Initialization. We assume that fragments in the spectrum obtained from the preprocessing step are indexed in some order. Each member of the population is a vector of fragment indexes representing a possible solution sequence to the problem. Given a vector $u[1 \dots m]$ of fragment indexes, the corresponding sequence is obtained by merging the fragments $F_{u[1]}, F_{u[2]}, \dots, F_{u[m]}$ in that order, where F_i is the ith fragment in the spectrum. Here, two adjacent fragments are put together by overlapping them as much as possible. We also maintain the constraint that each fragment index appears at most once in the encoding of a sequence. The vectors in the population can be of variable length. In what follows, we refer to each member of the population as a vector or sequence.

An initial population of size 120 is generated at random. The size of the population remains constant throughout the algorithm. The vectors in the initial population all have the same size, i.e., each vector has the same number of fragment indexes. However, since the fragments are of variable length after the preprocessing step, the corresponding sequences have different lengths. Suppose the spectrum obtained after preprocessing contains the fragments CTAGACG, TATCC, ACGTT, and the fragments are indexed in that ordered from 1 to 3. Then, the vector $(1, 3)$ yields the sequence CTAGACGTT, and the vector $(2, 1)$ yields the sequence TATCCTAGACG.

Fitness. The fitness of each sequence is calculated based on two factors: (i) the amount of overlap between adjacent fragments in the sequence, and (ii) the length of the sequence. The idea here is that the more overlap there are between adjacent fragments the shorter the sequence is. Also, if the length of the sequence is equal to $n+l-1$, where n is the cardinality of the spectrum and l is the length of each fragment, a bonus value is added to the value of the fitness. Note that $n + l - 1$ is the optimal length for a sequence that includes all fragments in the spectrum. More formally, let $u[1 \ldots k]$ be a vector representing a sequence U in the population. The fitness of U is defined as follows.

$$f(U) = \sum_{i=1}^{k-1} |F_{u[i]} \cap F_{u[i+1]}| + z$$

where z is the bonus value defined by

$$z = \begin{cases} s, & \text{if } |U| = n + l - 1, \\ s/||U| - (n + l - 1)|, & \text{otherwise.} \end{cases}$$

For our experiment, s was set to 100. The fitness can be computed efficiently using dynamic programming technique.

Parent Selection. The parents are selected using the standard proportional selection method where sequences that have higher fitness have a better chance of being selected. The standard roulette wheel scheme is used in our algorithm [13].

Crossover. We use a 3-point crossover in our algorithm. The offspring is constructed by selecting alternately from each parent after 3 cutpoints have been determined. Note that the members of the population are vectors of fragment indexes. This process may create offsprings that contain duplicated fragment indexes. A repair algorithm is used to get rid of any repeated fragments and to ensure that each fragment appears at most once within the offspring. The repair algorithm works by replacing the repeated fragments with fragments from the spectrum that are not currently in use by the sequence.

Mutation. The mutation is performed on each newly created offspring sequence as follows. With a probability of 10% each fragment index in the offspring is swapped with another randomly selected fragment index.

Local Optimization. The local optimization algorithm has two steps. The first step is to scan the sequence sequentially and identify a pair of adjacent fragments, say x and y, that has the smallest overlap. Find the fragment, say z, that has the highest overlap with x. Replace y with z. The vector is then repaired, if needed, to eliminate duplicated fragments.

The second step is to rearrange the fragments in the sequence in the hope of improving its fitness value. This is done by first finding the two pairs of adjacent fragments that have the two smallest overlaps. Let s, t be the first pair and x, y be the second pair. That is, assume that the vector $u = (a, \ldots, s, t, \ldots, x, y, \ldots z)$. We then construct a new vector u' by swapping the fragments between t and x with the fragments from y to the end of u. Thus, $u' = (a, \ldots, s, y, \ldots, z, t, \ldots, x)$. If the fitness of u' is better than that of u, we replace u by u'. Otherwise, we keep u and discard u'. By using dynamic programming technique the local optimization algorithm and the repair algorithm can be efficiently implemented.

Replacement Scheme. If the fitness of the new offspring is larger than the fitness of the worse of the two parents, then we replace that parent with the new offspring. Otherwise, we discard the new offspring.

Stopping Condition. The algorithm terminates if there is no improvement in the total fitness of the population in 400 consecutive generations, or if the number of generations exceeds 50,000.

4 Experimental Results

In this section we first describe the performance of our algorithm in comparison with some existing algorithms for the DNA sequencing problem. We then show the result of our algorithm on an extensive set of data generated by us using genome sequences from the GenBank (www.ncbi.nlm.nih.gov/Genbank/). Our algorithm was implemented in C++ and was run on a PC with Pentium IV 2.4GHz Intel processor with 512MB of RAM.

Our first set of test data is from [7][15]. We used it to compare our algorithm against the Hybrid Genetic Algorithm of [7] and the Tabu Search algorithm in [1]. The data from this set consists of spectra having 100, 200, 300, 400 and 500 fragments. There are 40 spectra for each size, for a total of 200 instances. The fragment length in all of these instances is 10. In each instance there are 20% positive errors and 20% negative errors.

To determine the quality of the solution we follow [7] and use the classical pairwise Smith-Waterman sequence alignment algorithm to compare the solution output by the algorithm with the known sequence from which the spectrum was derived. We use two values from the output of the Smith-Waterman algorithm: the match percentage and the similarity score. In addition, as in [7], for each instance tested we include the number of times the algorithm finds the optimal answer. This number is called the optimum number. Table 1 and Figure 2 summarize the results of the comparison. It can be seen that our algorithm performs significantly better than the other algorithms as the sequence length gets larger. More details can be seen in Figures 2 (a), (b) and (c).

Even though the running times are available for all algorithms, we cannot compare the running times, since different machines were used. For our algorithm, the average running time was from 0.6 second for the smallest problem size to 15.1 seconds for the largest problem size tested. The Hybrid GA and Tabu Search algorithms were run on a PC with a Pentium II 300MHz processor and 256MB of RAM. The average running times range from 13.5 seconds to 437.9 seconds for the Hybrid GA and from 14.1 seconds to 471.5 seconds for the Tabu Search algorithm. More details can be found in Table 1.

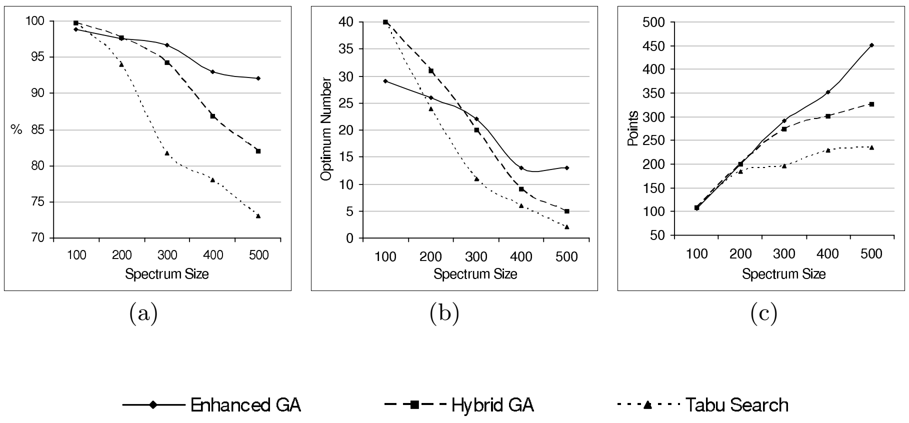

Fig. 2. Comparison between our algorithm Enhanced GA, Hybrid GA and Tabu Search: (a) Match Percentage, (b) Optimum Number, and (c) Similarity Score

The second set of test data we used was generated by us using three genomes obtained from the GenBank. Table 2 shows the details of the genomes that we used. For each of the three genomes we generated 10 sequences of each length in the set $\{100, 200, 300, 400, 500, 1000, 2000\}$. That is, for each genome we generated 70 sequences. From each of these sequences we generated spectra with fragments of length 10 and 20. For each fragment length, we generated spectra with 13 different combinations of positive and negative errors, ranging from 0 to 20% errors. Hence, for all three genomes we generated a total of $3 \times 70 \times 2 \times 13 = 5,460$ spectra. For each input spectrum the algorithm was run 50 times. We used the Mersenne Twister random number generator of [17] in our algorithm.

We have also generated a similar set of spectra with fragments of length 50. The algorithm was tested on all spectra of three different lengths: 10, 20, and 50. We observe that the longer the fragments are, the better the results are. In fact, with fragment length of 50, our algorithm almost always found the optimal answers, and thus, we do not include the data for fragments of length 50 here. We used different fragment lengths since in practice different hybridization

Table 1. Summary of results by Enhanced GA, Hybrid GA and Tabu Search

Algorithm		100	200	300	400	500
Enhanced GA	Average Similarity Score (pt)	106.7	200.6	291.6	352.2	451.6
(Pentium IV,	Average Similarity Score (%)	98.9	97.6	96.7	92.9	92.0
2.4GHz,	Running time (sec)	0.6	1.5	3.6	8.6	15.1
512MB RAM)	Optimum no.	29	26	22	13	13
HGA	Average Similarity Score (pt)	108.4	199.3	274.1	301.7	326.0
(Pentium II,	Average Similarity Score (%)	99.7	97.7	94.3	86.9	82.0
300MHz,	Running time (sec)	13.5	63.4	154.9	263.4	437.9
256MB RAM)	Optimum no.	40	31	20	9	5
Tabu Search	Average Similarity Score (pt)	108.4	184.1	196.6	229.5	235.1
(Pentium II,	Average Similarity Score (%)	99.7	94.0	81.8	78.1	73.1
300MHz,	Running time (sec)	14.1	60.8	177.7	258.3	471.5
256MB RAM)	Optimum no.	40	24	11	6	2

Table 2. Genomes from the GenBank used in testing the Genetic Algorithm

Sequence	Length(bp)
Human immunodeficiency virus 2	10,359
Drosophila melanogaster DNA sequence of white locus	14,245
Canis familiaris clone RP81-60B6	165,116

techniques may require different fragment lengths. Normally, hybridization rate is better if the fragment length is larger. However, for *in situ* hybridization small fragment length is required [18].

As in the case of the first data set, we use the Smith-Waterman sequence alignment algorithm to determine the quality of the solutions returned by the algorithm. We used an implementation of the Smith-Waterman algorithm provided by Jie Li of the Iowa State University [16]. Figures 3 and 4 show the performance and running time of our algorithm on the second set of data. In Figure 3, the left graph shows the match percentage for spectra with fragment length 10, and the right graph is for spectra with fragment length 20. The x-axis shows the various error combinations in the input spectrum. The notation -a+b indicates spectra with a% negative error and %b positive error. For each error combination, the match percentage shown is the average of the match percentages taken from all spectra generated from the three genomes and with 50 runs for each such spectrum. In all cases, the match percentage is over 85%. It can be observed that spectra with longer fragment length seem to be easier to solve than ones with smaller fragment length. The same machine that we used to test the algorithm on the first data set was used for the second data set. Figure 4 shows that spectra with higher error percentage seem to take longer than spectra with smaller error percentage.

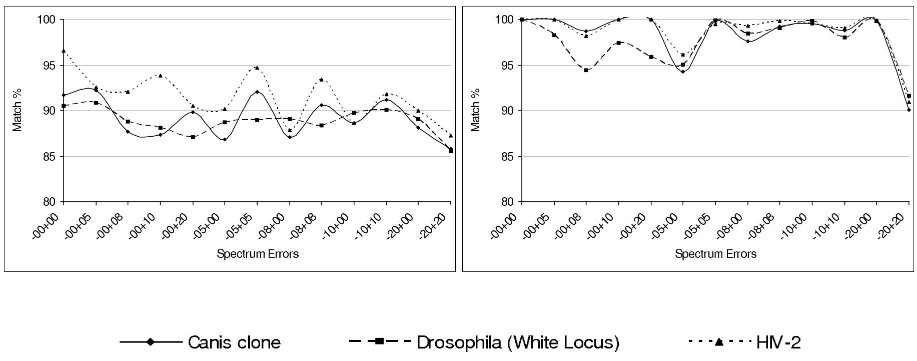

Fig. 3. Plot of match percentage of Enhanced GA against error combination for spectra with fragment length of 10 (left) and 20 (right). The label -a+b indicates spectra with a% negative error and %b positive error.

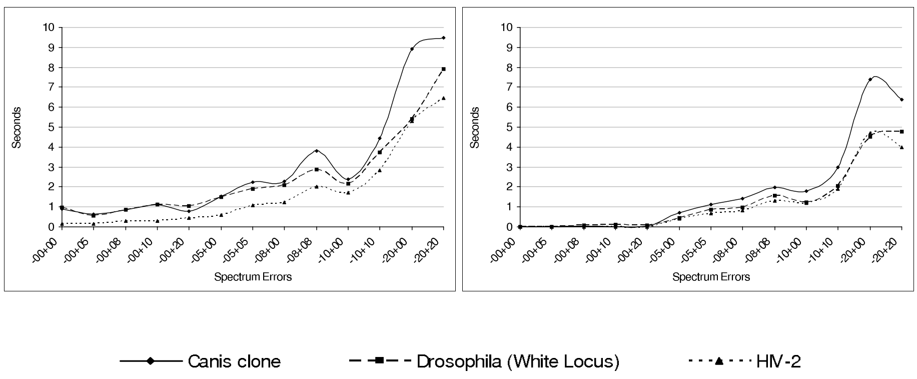

Fig. 4. Plot of running time of Enhanced GA against error combination for spectra with fragment length of 10 (left) and 20 (right). The label -a+b indicates spectra with a% negative error and %b positive error.

Experimental results from the two data sets suggest that our algorithm performs very well against existing algorithms. It is also very robust against different combinations of errors.

5 Conclusion

This paper introduced a new enhanced genetic algorithm for the DNA Sequencing problem. The results produced by the algorithm were very good and in many cases were optimal or close to optimal and were frequently better than existing algorithms. Taking into account the difference in speed of the machines on which the various algorithms were run, our algorithm seems to be comparable if not faster than existing algorithms. This paper did not look at the case when the input spectrum contains repeated fragments. Repeated fragments can cause the algorithm not to be able to find optimal answers even when there are no other types of errors in the spectrum. This type of error diminishes if the fragment length increases. We plan to look into the problem of dealing with repeated fragments.

Acknowledgements. The authors would like to thank Jacek Blazewicz and Marta Kasprzak for providing them with the data used in [7], Steven S. Skiena for useful discussions, and Jie Li for providing them with an implementation of the Smith-Waterman algorithm. The authors would also like to thank Erick Cantu-Paz and the anonymous referees for helpful comments.

References

1. Blazewicz, J., P. Formanowicz, F. Glover, M. Kasprzak and J. Weglarz, "An Improved Tabu Search Algorithm for DNA Sequencing with Errors," Proceedings of the III Metaheuristics International Conference (MIC), 1999, pp. 69–75.
2. Blazewicz, J., P. Formanowicz, M. Kasprzak, W.T. Markiewicz and J.Weglarz, "DNA Sequencing with Positive and Negative Errors," Journal of Computational Biology 6, 1999, pp. 113–123.
3. Blazewicz, J., P. Formanowicz, M. Kasprzak, W. T. Markiewicz and J. Weglarz, "Tabu Search for DNA Sequencing with False Negatives and False Positives," European Journal of Operational Research 125, 2000, pp. 257–265.
4. Blazewicz, J. and M. Kasprzak, "Complexity of DNA sequencing by hybridization," Theoretical Computer Science, 290, 2003, pp. 1459-1473.
5. Blazewicz, J., A. Kaczmarek, M. Kasprzak, W. T. Markiewicz and J. Weglarz, "Sequential and Parallel Algorithms for DNA Sequencing," Computer Applications in the Biosciences 13, 1997, pp. 151–158.
6. Blazewicz, J., J. Kaczmarek, M. Kasprzak, J. Weglarz and W. T. Markiewicz, "Sequential Algorithms for DNA Sequencing," Computational Methods in Science and Technology, 1, 1996, pp. 31–42.
7. Blazewicz, J., M. Kasprzak and W. Kuroczycki, "Hybrid Genetic Algorithm for DNA Sequencing with Errors," Journal of Heuristics, 8, 2002, pp. 495–502.
8. Blum, A., T. Jiang, M. Li, J. Tromp and M. Yannakakis, "Linear Approximation of Shortest Superstrings," Journal of the ACM, 41(4), 1994, pp. 630–647.
9. Cummings, M. R. *Human Heredity: Principles and Issues,* West Publishing Company, 1991.
10. Fogel, G. B. and K. Chellapilla, "Simulated Sequencing by Hybridization Using Evolutionary Programming," Proc. of the IEEE Congress on Evolutionary Computation, CEC'99, 1999, pp. 445–452.
11. Fogel, G. B., K. Chellapilla and D. B. Fogel, "Reconstruction of DNA Sequence Information From a Simulated DNA Chip Using Evolutionary Programming," Lecture Notes in Computer Science, edited by V. W. Porto, N. Saravanan, D. Waagen and A. E. Eiben, Vol. 1447, 1998, pp. 429–436.
12. Frieze, A. and W. Szpankowski, "Greedy Algorithms for the Shortest Common Superstring That Are Asymptotically Optimal," Algorithmica, 21(1), 1998, pp. 21–36.

13. Goldberg, D. E., *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley, 1989.
14. Haan, N. M. and S. J. Godsill, "Sequential Methods For DNA Sequencing" Department of Engineering, University of Cambridge, U.K.
15. Kasprzak, M., Personal communications, August 2003.
16. Li, J. "Implementation of Smith-Water Alignment Algorithm," Iowa State University, Personal Communication.
17. Matsumoto, M. and T. Nishimura, "Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator," ACM Transactions on Modeling and Computer Simulation, 8(1), January 1998, pp. 3–30.
18. *Nonradioactive In Situ Hybridization Application Manual,* Technical Manual, Roche Applied Science.
19. Percus, A. G. and D. C. Torneyy, "Greedy Algorithms for Optimized DNA Sequencing," Los Alamos National Laboratory, Los Alamos, NM 87545.
20. Pevzner, P. A., *Computational Molecular Biology, An Algorithmic Approach,* The MIT Press, second printing 2001, Chapter 4, pp. 59-63.
21. Pevzner, P. A., H. Tang and M. S. Waterman, "An Eulerian Path Approach to DNA Fragment Assembly," Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA; and Departments of Mathematics and Biological Sciences, University of Southern California, Los Angeles, CA, June 7, 2001.
22. Phan, V. T. and S. Skiena, "Dealing with Errors in Interactive Sequencing by Hybridization," Oxford University Press, 17(10), 2002, pp. 1-9.
23. Skiena, S., Personal communications, September 2003.
24. Waterman, M.S., *Introduction to Computational Biology: Maps, Sequences and Genomes,* Chapman & Hall, London, 1995.