Evolutionary Sequence Generation for Reliable DNA Computing

Soo-Yong Shin, Dong-Min Kim, In-Hee Lee, Byoung-Tak Zhang
Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National University
Seoul 151-742, Korea
{syshin, dmkim, ihlee, btzhang}@bi.snu.ac.kr

Abstract - Since DNA computing technologies use the bio-molecules as basic computing materials, DNA computing involves the possibilities for errors caused by the chemical characteristics of bio-molecules. To overcome these drawbacks, Many researchers have studied the design of DNA sequences to reduce the possibilities for illegal reactions. In this paper, we developed an evolutionary sequence generation system to minimize the potential errors in DNA sequences for reliable DNA computing. We verified our system by investigating the sequences designed by another sequence generator, and generated the sequences for solving travelling salesman problems.

I. Introduction

DNA computing is the computing model that considers DNA molecules as computing or storage materials and biological laboratory experiments as operators. However, DNA computing involves some problems due to the nature of chemical characteristics of bio-molecules. Generally chemical reactions are known as very reliable mechanisms, but often they are not satisfactory for computing purpose because computing devices should operate without error. Therefore the realizations of DNA computing applications are influenced by the reliability of the reactions of DNA.

Various approaches have been proposed to overcome these drawbacks, and most of the approaches focus on the design of DNA sequences to reduce the possibilities for illegal reactions. If the potential of errors that can occur in the biological reactions are removed in advance when generating DNA sequences, more efficient and reliable results can be achieved. In this paper, we propose to use evolutionary algorithms to generate error-preventing sequences.

Based on previous works[1][2], we improve the DNA sequence generation system as a part of DNA computing simulator, called NACST (Nucleic Acid Computing Simulation Toolkit). If the criteria of DNA sequence fitness are well determined, sequence design problem can be transformed to an optimization problem in combination with determined fitness measures such as multiobjective optimization problems. Thus we describe well-known fitness measures by a numerical expression. Then we use evolutionary algorithm to solve the sequence optimization problems.

The paper is organized as follows. In Section 2, the previous works of DNA sequence design are presented. Section 3 describes the proposed evolutionary sequence generation system. Section 4 and 5 provide the experimental results and conclusions.

II. Previous Works

Many researchers have proposed various algorithms and methods to design error-preventing DNA sequences for DNA computing. Deaton and Garzon identified the types of errors that can lead to false positives in Adleman's original techniques[3]. They also gave a theoretical bound on the size of problems that can be solved reliably[4]. They introduced a new measure of hybridization likelihood based on Hamming distance and proposed a theory of error-preventing codes for DNA computing[5].

Marathe et al. used dynamic programming approach to design DNA sequences based on Hamming distance and free energy[6]. Frutos et al. proposed a template-map strategy to select a huge number of dissimilar sequences by using only a significantly smaller number of templates and maps[7]. Hartemink et al. implemented the program "SCAN" to generate sequences for the programmed mutagenesis[8]. Feldkamp et al. also proposed the sequence construction system, DNASequenceGenerator[9]. It uses a directed graph, where the nodes are base strands and the successors of a node are the four strands that can appear as successors in a longer sequence.

Evolutionary algorithms have also been used for sequence design. Deaton et al. used genetic search for good encodings[4], Arita et al. developed the DNA sequence design system using genetic algorithm and random generate-and-test algorithm[10]. Tanaka et

0-7803-7282-4/02/$10.00 ©2002 IEEE

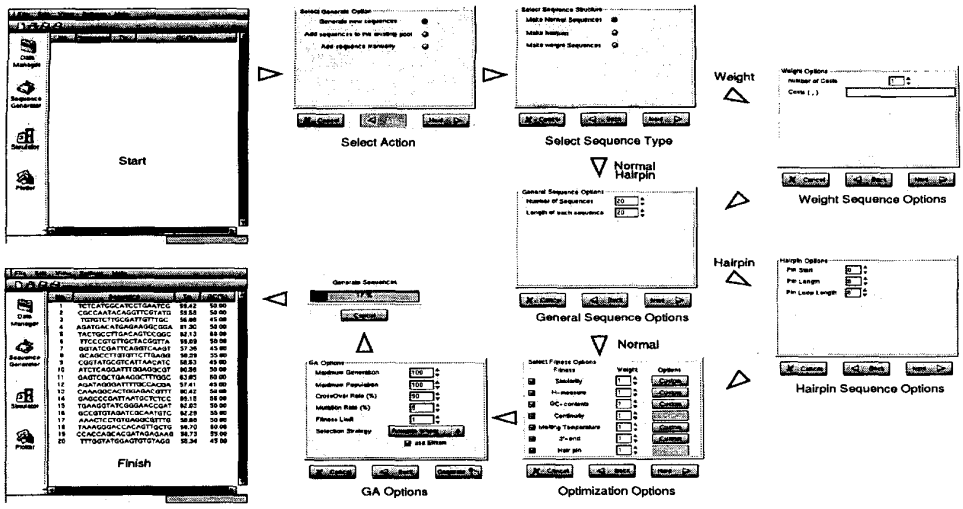

Fig. 1. Sequence generation process

al. listed up some sequence fitness criteria, and they generated the sequence with simulated annealing technique[11]. Reben et al. developed "PUNCH" to optimize DNA sequences using genetic algorithms[12].

III. Evolutionary Sequence Generation

We developed the system NACST (Nucleic Acid Computing Simulation Toolkit) for DNA sequence design and DNA computing simulation. NACST is developed for reliable and efficient DNA computing[1][2][13].

NACST consists of two main parts. One is the DNA sequence generation part which designs error-preventing sequences to reduce the potential of errors in DNA sequences while the chemical reactions occur. The other is the DNA computing simulation part that simulates the experiments such as ligation, PCR, gel electrophoresis, and so on. The simulation part is used to develop or test new molecular algorithms or computing models. Additional parts include a sequence test part, a data manager part, and a plotter part. The sequence test part examines the input sequences to evaluate the fitness of the given sequences. The plotter part plots a necessary graph such as fitness evaluation process, and the data manager part is a tool for handling input data, e.g., graph representation for Hamiltonian path problems or the genetic algorithm parameters. The GUI of NACST is shown in Fig. 1. We will focus on the sequence generation part in this paper.

The sequence generation part is divided into two subparts such as making new sequences and adding sequences to an existing sequence pool. We can add the existing DNA sequences to the sequence pool, or we can generate the new sequences and then add these sequences to the pool. When generating the sequences, we can control the various sequence properties, for example GC contents, self complementary structure such as hairpin and internal loops, melting temperature, sequence similarities, and so on. In addition, we can test the fitness of the existing sequences. The processes of generating sequences and the various sequence generation option windows are shown in Fig. 1. First, the action types, e.g., generating new sequences or adding the sequences to a pool, are selected. Next, the sequence type such as normal or hairpin sequences is chosen. Then the number and the length of the sequences are decided, and the fitness measures options plus genetic algorithm parameters are adjusted. After generating the sequences, the main window shows the result sequences with their Tm and GC contents.

As mentioned in the previous section, DNA sequence design is considered as an optimization problem. More precisely, since the fitness measures of sequence generation are multiple, conflicting objectives, DNA sequence design is transformed to multiobjective optimization. So an evolutionary algorithm is suitable for sequence design problems. As a first step, we use the conventional single-objective genetic algorithm with multiple point crossover and single point mutation. Therefore, the fitness function is the weighted sum of required fitness measures:

$$fitness = \sum_i w_i f_i. \quad (1)$$

For simplicity, we set each weight to one. The best fitness value is zero, therefore we minimize Eqn. (1). The following subsection will describe the fitness measures for sequence optimization.

A. fitness measures

Based on the evaluation terms listed up by Tanaka et al. [11], we describe 7 fitness evaluation functions more precisely. We define an alphabet consisted of each single nucleotide as $\Lambda = \{A, C, G, T\}$, then the set of all DNA sequences is denoted as Λ^*. Let $a, b \in \Lambda$ and $x, y \in \Lambda^*$. And the length of sequence x is denoted as $|x|$.

A.1 H-measure

H-measure is an important factor for preventing the mismatches such as shifted mismatch and mismatched hybridization[5]. In our implementation, H-measure considers both continuous and discontinuous undesired matches. And the user can set the threshold for allowed continuous and discontinuous matches. Thus, the H-measure function of two sequences x and y is

$$H(x,y) = \sum_{k=-|x|}^{|y|} \begin{pmatrix} I(D(y, s_k(x)), T_h) \\ + D_{run}(y, s_k(x)) \end{pmatrix}, \quad (2)$$

$$I(X,t) = \begin{cases} X & \text{if } X \text{ is larger than threshold } t \\ 0 & \text{otherwise} \end{cases},$$

where $s_k(*)$ denotes k right (left) shift in the case of $k > 0$ ($k < 0$), $I(*,*)$ the threshold function, $D(*,*)$ the total number of unexpected matches, T_h user-defined threshold for H-measure, and $D_{run}(*,*)$ denotes the sum of unexpected continuous matches which is larger than user defined threshold. Therefore the fitness function of H-measure is

$$f_{H-measure} = \sum_{i<j} H(x_i, x_j). \quad (3)$$

A.2 3'-end Hybridization

Preventing hybridization at 3'-end is important since it can cause unexpected extension during PCR. It is the same as H-measure except that it considers only user defined length l at 3'-end of each sequence.

$$H_{end}(x,y) = \sum_{k=-|e_l(x)|}^{|e_l(y)|} \begin{pmatrix} I(D(e_l(y), s_k(e_l(x))), T_e) \\ + D_{run}(e_l(y), s_k(e_l(x))) \end{pmatrix}, \quad (4)$$

where $e_l(x)$ denotes the subsequence of length l and 3'-end of x, T_e denotes threshold for H_{end}. The fitness function is

$$f_{3'-end} = \sum_{i<j} H_{end}(x_i, x_j). \quad (5)$$

A.3 Similarity

Similarity measure is necessary to prevent undesired hybridization by keeping the sequences as unique as possible. Similarity also considers both continuous and discontinuous coincidences between two sequences with a shift.

$$S(x,y) = \sum_{k=-|x|}^{|y|} \begin{pmatrix} I(d(y, s_k(x)), T_s) \\ + d_{run}(y, s_k(x)) \end{pmatrix}, \quad (6)$$

where $d(*,*)$ denotes the total number of coincidences, T_s threshold for similarity, and $d_{run}(*,*)$ denotes the sum of continuous coincidences which is larger than user defined threshold. Thus the fitness function is

$$f_{Similarity} = \sum_{i<j} S(x_i, x_j). \quad (7)$$

A.4 Continuity

If the same bases occur continuously in a sequence, the resulting sequence can have an unexpected 3-dimensional structure. The fitness measure for continuity is

$$C(x) = \sum_{a \in \Lambda} C_a(x), \quad (8)$$

where $C_a(x)$ denotes the sum of the square of continuous occurrence of base a in sequence x that is larger than user defined threshold. The fitness function is

$$f_{Continuity} = \sum_{i} C(x_i). \quad (9)$$

A.5 Hairpin

Generally the hairpin structure formation is not desirable, but it is possible that we may need it in some cause. Thus we made two variants of hairpin fitness measure, one for preventing the hairpin structure and the other for making planned hairpin structure. We consider a hairpin formation as the sequence x folded with cth base by a center, and the number of self-hybridized matches as the cost for this hairpin. The hairpin can be formulated as

$$HP(x) = \sum_{r=5}^{|x|-2 \cdot pinlen} \sum_{c=pinlen+\lceil r/2 \rceil}^{|x|-pinlen-\lfloor r/2 \rfloor} Hairpin(x,c), \quad (10)$$

where $Hairpin(x,c)$ denotes the number of matches when sequence x is folded around cth base, *pinlen* denotes the minimum sequence length of hybridization to form hairpin. Therefore the fitness function for generating normal linear sequence is

$$f_{Hairpin} = \sum_{i} HP(x_i). \quad (11)$$

TABLE I

FITNESS OF THE CODES IN DEATON'S PAPER.

Sequence ($5' \rightarrow 3'$)	H-measure	3'-end	Similarity	Continuity	Hairpin	GC%	Tm
Good codes							
CTTGTGACCGCTTCTGGGGA	82	21	34	16	3	60	62.83
CATTGGCGGCGCGTAGGCTT	50	49	15	0	3	65	71.06
ATAGAGTGGATAGTTCTGGG	19	14	41	0	6	45	54.90
GATGGTGCTTAGAGAAGTGG	34	9	20	0	0	50	57.50
TGTATCTCGTTTTAACATCC	46	3	5	16	11	35	50.59
GAAAAAGGACCAAAAGAGAG	19	2	2	41	0	40	54.89
TTGTAAGCCTACTGCGTGAC	-	-	-	0	6	60	58.87
Bad codes							
ATCAGCTGGATTCATCTGAA	179	95	177	0	9	40	55.94
ATCAACAGAAATCCGCGGAA	139	67	74	9	6	45	61.64
ATCAGCTGAGGTCTGGTGAG	102	10	87	0	12	55	59.78
GTCCGCTGTATTCTCGTGAT	48	11	32	0	0	50	59.24
TTCAACTGTTTTCAGCTGTG	51	14	38	16	10	40	52.69
TTCACCTTTATTGAGCCGAA	29	12	20	9	9	40	53.83
TTCAGCCGATTTGCGGAGAA	-	-	-	9	19	50	61.06

When generating a hairpin structure, we gave the penalty for both an undesired hairpin formation and an imperfect hybridization at the planned position. Thus the fitness function for generating hairpin structure is

$$f_{Hairpin} = \sum_i (HP(x_i) + HP_{desired}(i)), \quad (12)$$

$$HP_{desired}(i) = \begin{cases} 0 & \text{if desired hairpin formed} \\ pinlen & \text{otherwise} \end{cases},$$

where $HP(x_i)$ is excluded when r and c are the values for desired hairpin.

A.6 GC Contents

GC contents affect the chemical properties of DNA sequences. The fitness function is

$$f_{GCcontent} = \sum_i (GC_{target}(x_i) - GC_{generated}(x_i))^2, \quad (13)$$

where $GC_{target}(x)$ is the target GC contents of sequence x, and $GC_{generated}(x)$ is the real GC contents.

A.7 Melting Temperature

Melting temperature is also important for efficiency and reliability of the DNA reaction. There are many equations to calculate melting temperature such as the Wallace 2-4 rule, the GC% method, and the nearest-neighbor model. We use the nearest neighbor model (NN) with SantaLucia's unified NN parameters[14] to calculate melting temperature and GC% method.

$$f_{Tm} = \sum_i (Tm_{target}(x_i) - Tm_{generated}(x_i))^2. \quad (14)$$

IV. Experimental Results

To verify our approach, we test the sequences in [3] by the NACST sequence test part. In [3], genetic algorithm was used to design good codewords for Adleman's graph, and the Hamming bound is used to be an upper bound on the number of reliable encodings. The test results are shown in Table I. Originally the good encodings all satisfied the minimum distance properties dictated by the Hamming bound, and the bad encodings did not.

For the parameters of testing, H-measure for continuous base pairing is 5, discontinuous case is 16%, 3'-end is same as H-measure but only considers 10 bp in the 3'-end, similarity is 16%, continuity is 2, Hairpin can be formed by 3 base pairing and 5 base loop, and Tm is decided by NN method with $1M$ salt concentration and $10nM$ DNA concentration. This means that we give penalties in H-measure from 6 base continuous hybridization, over 16% hybridization in the whole sequence, hairpin formation, and so on. As explained in the previous section, while continuity, hairpin, GC contents, and Tm are evaluated by each sequence, H-measure, 3'-end, and similarity are compared with other sequences. Therefore, the values of H-measure, 3'-end, and similarity decreases as the index goes on, and the last index sequence ("TTGTAAGCCTACTGCGTGAC" in Table I) does not have any value, since no other sequence is left to be compared.

As shown in Table I, the good codes give better results than the bad codes. The good codes have smaller H-measure values, 3'-end values, and similarity values than those of bad codes. It implies that the good sequences may not hybridize with the wrong sequences.

TABLE II

GENERATED VERTEX SEQUENCES AND WEIGHT SEQUENCES FOR TSP.

Vertex sequences								
No.	Sequence ($5' \rightarrow 3'$)	H-measure	3'-end	Similarity	Continuity	Hairpin	GC%	Tm
0	AGGCGAGTATGGGGTATATC	68	13	19	16	0	50	60.73
1	CCTGTCAACATTGACGCTCA	48	19	46	0	3	50	59.24
2	TTATGATTCCACTGGCGCTC	30	5	51	0	0	50	59.00
3	ATCGTACTCATGGTCCCTAC	27	4	23	9	6	50	56.81
4	CGCTCCATCCTTGATCGTTT	15	14	4	0	7	50	58.13
5	CTTCGCTGCTGATAACCTCA	13	4	4	0	6	50	59.44
6	GAGTTAGATGTCACGTCACG	-	-	-	0	9	50	56.97
Weight sequences								
Edge cost	Sequence ($5' \rightarrow 3'$)	H-measure	3'-end	Similarity	Continuity	Hairpin	GC%	Tm
3	ATGATAGATATGTAGATTCC	6	4	4	0	3	30	47.89
5	GGATGTGATATCGTTCTTGT	1	3	2	0	3	40	54.62
7	GGATTAGCAGTGCCTCAGTT	0	2	3	0	3	50	58.37
9	TGGCCACGAAGCCTTCCGTT	1	1	0	0	0	60	64.51
11	GAGCTGGCTCCTCATCGCGC	-	-	-	0	13	70	68.88

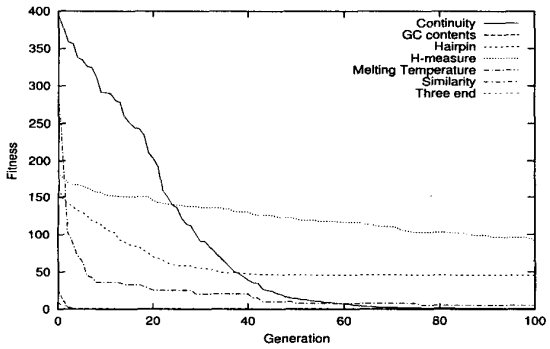

Fig. 2. best fitness vs. generation

Also the smaller hairpin values represent the prohibition of the illegal structures. However, good codes do not seem to be tractable sequences for real experiment. For example, the 3'-end value of the second sequence in Table I is much bigger than the others. By investigating the second sequence, we find that the second sequence "CATTGGCGGCGCGTAGGCTT" and the last sequence "TTGTAAGCCTACTGCGTGAC" can hybridize with each other through underlined sequences. We also find that there are many repeated 'A's in the sequence "GAAAAAGGACCAAAAGAGAG". Even the sequence "TTCACCTTTATTGAGCCGAA" has a possibility to form hairpin structure by the underlined sequences "TTC" and "GAA".

Next, we traced fitness evaluation process. We generate 10 sequences with 50% GC contents, Tm ranges from 60 to 70^oC, and the same parameter settings as described in the previous experiment. The population size is 100, maximum generation is 100, crossover rate is 0.9, and mutation rate is 0.05. The graph shows the average value of best fitness of 10 experiments. As depicted in the Fig. 2, most fitness measure can converge in the early generation; even GC contents and 3'-end drop to 0 before 4th generation. But, since H-measure does not converge well, we can say that H-measure is the most complicated fitness measure for DNA computing.

A. Travelling salesman problems

We generate DNA sequences for travelling salesman problems (TSP). The target TSP is a 7-nodes incomplete graph whose edge weights are 3, 5, 7, 9, and 11. Thus, 7 vertex sequences, 5 weight sequences, and 24 edge sequences are generated. The population size is 1000, maximum generation is 1000, crossover rate is 0.9, and mutation rate is 0.05. Edge sequences are composed of two subsequences. One is a linker sequence and the other is a weight sequence. The linker sequence is decided by vertex sequence the same way as Adleman's experiment. The weight sequence represent the edge costs by GC contents of the sequence, and is located in the middle of edge sequence. See [15] for more detailed description.

The generated vertex and weight sequences with their fitness are shown in Table II. GC contents of the weight sequences are 30, 40, 50, 60, and 70% for cost 3, 5, 7, 9, and 11 respectively. First the vertex sequences are generated, then the weight sequences are added to the sequence pool by evaluating fitness with the generated

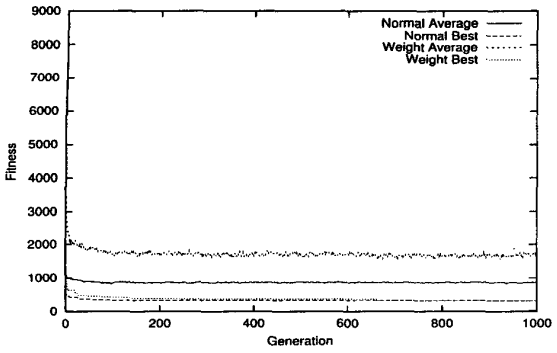

Fig. 3. fitness vs.generation for TSP. Normal means the vertex sequences and weight represents the weight sequences.

vertex sequences. In Comparison with Table I, our sequences show better fitness values. Especially with uniform GC contents (50%), we can design more feasible sequences to DNA computing experiments.

The fitness evaluation graph for TSP sequences is shown in Fig. 3. We can find the tractable sequences in the early generation. As we expected, the best fitness of vertex sequences and weight sequences are similar. But average fitness of weight is much larger than that of vertex, since the weight sequences evaluate with each other and also with vertex sequences.

V. Conclusions

We developed an evolutionary sequence generator as a part of DNA computing simulator, called NACST (Nucleic Acid Computing Simulation Toolkit). We considered sequence design as an optimization problem with weighted sum of necessary fitness measures, and used conventional evolutionary algorithm to generate the DNA sequences. We verified our method by investigating DNA sequences designed by another sequence generator, and generated tractable DNA sequences to solve travelling salesman problem.

The work in progress is to improve the evolutionary sequence generator to more reliable system using multi-objective evolutionary algorithms, and to build the simulation system to utilize massive parallelism of DNA computing and describe the thermodynamics of bio-molecules.

Acknowledgments

This research was supported in part by the Ministry of Education & Human Resources Development under the BK21-IT Program and the Ministry of Commerce, Industry and Energy through MEC project. The RIACT at Seoul National University provides research facilities for this study.

References

[1] B.-T. Zhang and S.-Y. Shin, "Molecular algorithms for efficient and reliable DNA computing," in *Proceedings of Genetic Programming 1998.* 1998, pp. 735–742, Morgan Kaufmann.

[2] B.-T. Zhang and S.-Y. Shin, "Code optimization for DNA computing of maximal cliques," in *Advances in Soft Computing - Engineering Design and Manufactoring.* 1999, Springer.

[3] R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, and S.E. Stevens Jr., "Good encodings for DNA-based solutions to combinatorial problems," in *Proceedings of the Second Annual Meeting on DNA Based Computers*, 1996, pp. 247–258.

[4] R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and Jr. S. E. Stevens, "Reliability and efficiency of a DNA-based computation," *Physical Review Letters*, vol. 80, no. 2, pp. 417–420, 1998.

[5] M. Garzon, P. Neathery, R. Deaton, R. C. Murphy, D. R. Franceschetti, and Jr. S. E. Stevens, "A new metric for DNA computing," in *Proceedings of Genetic Programming 1997.* 1997, pp. 472–478, The MIT Press.

[6] A. Marathe, A. E. Condon, and R. M. Corn, "On combinatorial DNA word design," in *Proceedings of 5th DIMACS Workshop on DNA Based Computers*, Erik Winfree and David K. Gifford, Eds. 1999, pp. 75–89, American Mathematical Society.

[7] A. G. Frutos, A. J. Thiel, A. E. Condon, L. M. Smith, and R. M. Corn, "DNA computing at surfaces : 4 base mismatch word design," in *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers*, 1997, p. 238.

[8] A. J. Hartemink, D. K. Gifford, and J. Khodor, "Automated constraint-based nucleotide sequence selection for DNA computation," in *Proceedings of 4th DIMACS Workshop on DNA Based Computers*, 1998, pp. 227–235.

[9] U. Feldkamp, S. Saghafi, W. Banzhaf, and H. Rauhe, "DNAsequencegenerator - a program for the construction of DNA sequences," In Jonoska and Seeman [16], pp. 179–188.

[10] M. Arita, A. Nishikawa, M. Hagiya, K. Komiya, H. Gouzu, and K. Sakamoto, "Improving sequence design for DNA computing," in *Proceedings of Genetic and Evolutionary Computation Conference 2000*, 2000, pp. 875–882.

[11] F. Tanaka, M. Nakatsugawa, M. Yamamoto, T. Shiba, and A. Ohuchi, "Developing supprot system for sequence design in DNA computing," In Jonoska and Seeman [16], pp. 340–349.

[12] A. J. Ruben, S. J. Freeland, and L. Landweber, "PUNCH: an evolutionary algorithm for optimizing bit set selestion," In Jonoska and Seeman [16], pp. 260–270.

[13] S.-Y. Shin, B.-T. Zhang, and S.-S. Jun, "Solving traveling salesman problems using molecular programming," in *Proceedings of Congerss on Evolutionary Computation 1999.* 1999, pp. 994–1000, IEEE Press, Washington D.C., USA.

[14] J. SantaLucia Jr., "A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics," *Proc. Natl. Acad. Sci. USA*, vol. 95, pp. 1460–1465, 1998.

[15] S.-Y. Shin, J.-Y. Lee, S. J. Augh, B.-T. Zhang, and T.-H. Park, "Molecular approach to solve travelling salesman problems with temperature gradient," Technical Report BI-2001-1, Seoul National University, Seoul, Korea, 2001.

[16] N. Jonoska and N. C. Seeman, Eds., *7th international Workshop on DNA-Based Computers, Tampa, U.S.A., 10-13 June 2001.* University of South Florida.