Code Optimization for DNA Computing of Maximal Cliques

Byoung-Tak Zhang and **Soo-Yong Shin**

Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering, Seoul National University
Seoul 151-742, Korea.
E-mail: {btzhang, syshin}@scai.snu.ac.kr

Keywords: DNA computing, maximal clique problem, DNA code design, evolutionary optimization, NP-complete problems.

Abstract

Since the Adleman's experimental demonstration of its feasibility, DNA computing has been applied to a number of combinatorial optimization problems. Several experimental results have shown that DNA strands can be used to compute solutions to NP-complete problems. Usually they employ random codes to represent candidate solutions in DNA. However, some codes have better error-tolerance than others since current bio-lab experiment techniques are involved with reaction errors. In this paper we present an evolutionary method for optimizing the DNA codes for solving a given problem. The method uses a genetic algorithm to find best codes by emulating chemical reaction processes before actual biological experiments begin. Simulations have been performed to solve maximum clique problems, an NP-complete search problem. The results show that the optimized codes improve the reliability of DNA computing and reduce time and costs that may be caused by repeated bio-experiments.

1 Introduction

Due to advances in molecular biology it is nowadays possible to create a soup of roughly 10^{18} DNA strands that fits in a test tube. Adleman [1] has shown that each DNA strand can be used to compute solutions to an instance of the NP-complete Hamiltonian path problem (HPP). Lipton [6] came up with using DNA computing to solve the satisfiability (SAT) problem: given a propositional formula decide if it is satisfiable. Recently, Ouyang et al. [7] provided further evidence for the ability of DNA computing to solve NP-complete search problems. They solved a maximal clique problem (MCP) by means of molecular biology techniques.

Both Adleman and Lipton used random encodings in their study. It was, however, suggested that as the size of the problem grows, particular attention must be paid to errors and the formation of pseudopaths. In previous work [9] we studied an evolutionary method for further improvement of the efficiency and robustness of Adleman-style DNA computing by optimizing DNA encodings for representing problem instances. In this paper we extend this framework to solve a different combinatorial optimization problem, i.e. MCP, using a different DNA computing method suggested by Ouyang et al. [7].

The effectiveness of the proposed method is shown by simulations performed on two instances of the maximal clique problem. Our results indicate that using genetic algorithms to optimize the DNA codes improves robustness and fidelity of DNA computing. As a byproduct of our study in this paper we found that the new encoding scheme which uses restriction enzymes is in general better than the Adleman-style encoding in its fidelity.

The paper is organized as follows. In Section 2 we define the maximal clique problem and outline the algorithm to solve it by DNA computing. Section 3 describes the method of simulated DNA computing and its results. Section 4 presents the method for optimizing DNA codes their experimental results. Section 5 discusses the results.

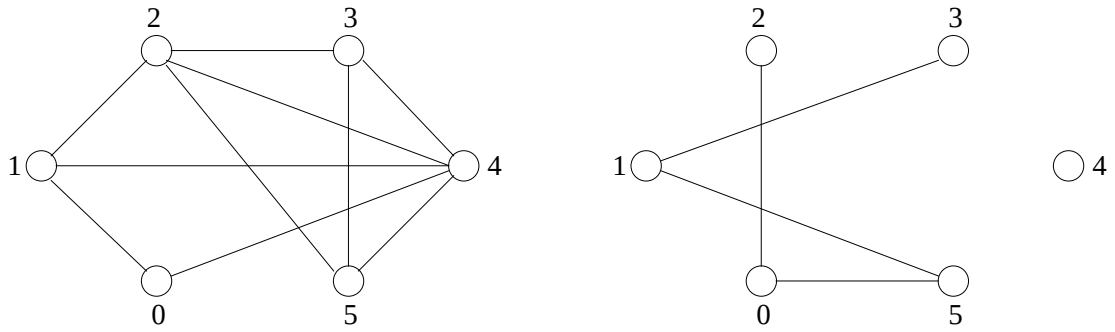

Figure 1: The maximum clique problem: (left) graph with 6 vertices, (right) its complementary graph.

2 DNA Solution of the Maximal Clique Problem

A clique is defined as a set of vertices in which every vertex is connected to every other vertex by an edge. The maximal clique problem is to find the largest clique, given a network containing N vertices and M edges. The graph of six vertices and 11 edges in Figure 1(left) defines such a problem. The vertices (5, 4, 3, 2) form the largest clique and thus the size of the largest clique in this network is four. Finding the largest clique has been proven to be an NP-complete problem [3].

Recently, Ouyang et al. [7] have solved this problem by means of molecular biology techniques. Their experiment proceeds as follows.

1. For a graph with N vertices, each possible clique is represented by an N-digit binary number. A bit set to 1 represents a vertex in the graph, and a bit set to 0 represents a vertex out of the clique. For example, the clique (4,1,0) is represented by the binary number 010011. In this way, the complete set of possible cliques is transformed into an ensemble of all N-digit binary numbers. This is called the complete data pool.

2. Pairs of vertices in the graph are found that are not connected by an edge. The graph containing all edges missing in the original graph is called the complementary graph (Figure 1(right)). Any two vertices connected in the complementary graph are disconnected in the original graph (Figure 1(left)) and therefore cannot be members of the same clique; this means that the corresponding bits cannot both be set to 1.

3. All numbers containing connections in the complementary graph are then eliminated from the complete data pool. For the problem in Figure 1, the numbers $***1*1$, $1****1$, $1***1*$, and $**1*1*$ are removed ($*$ can be either 1 or 0). For the problem in Figure 4, the numbers $***1*1$, $1****1$, $**1*1*$, $**11**$, $*11***$ are removed. The remainder of the data pool corresponds to all cliques in the original graph.

4. The remaining data pool was sorted to find the data containing the largest number of 1's. Each of these ones represents a vertex in the corresponding clique; therefore, the clique with the largest number of 1's tells us the size of the maximal clique.

The possible solutions are encoded as double-stranded DNA molecules. Each bit in a binary number is represented by two DNA sections corresponding to the bit's value (V_i) and its position (P_i). For a DNA molecule representing a six-digit binary number, there are six value sections (V_0 to V_5) sandwiched sequentially between seven position sections (P_0 to P_6) (see Figure 2). The last position section, P_6, is needed for polymerase chain reaction (PCR) amplification. Ouyang et al. report that P_i with a length of 20 base pairs worked well. The length of V_i was set to 0 bp if the value of $V_i = 1$, and to 10 bp

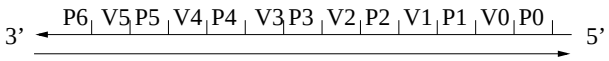

Figure 2: Encoding data in DNA.

Index	Position Sequence	Value Sequence
0	CGTAGAATTCTGCGAACCTT	GACGCGGCAG
1	AAGAGTCACCTATCAGTAAG	TTCATAACCG
2	CTTCCTGAAGAGAGATTACT	CCGGTCACTT
3	AGTGCTAACAAGCCTGCGCA	CCTGACGATT
4	TGCAGTCTTTGAGATAAAGG	AAAAACCCAC
5	CCTACGTGTGAAATAGACTC	CCACCGATCT
6	GAGAGGGGCGGGATCCAGGG	

Table 1: Parameters for the experiments.

if the value if $V_i = 0$. Therefore, the longest DNA has 200 bp corresponding to the number 000000, and the shortest DNA has 140 bp corresponding to the number of 111111. Each oligonucleotide consists of two position motifs and one value motif, $P_iV_iP_{i+1}$ for even i and complementary sequence $P_{i+1}V_iP_i$ for odd i. The data pool was digested with restriction enzymes. These enzymes break DNA at specific restriction sites, which were embedded within the sequences for $V_i = 1$. The broken strings were not amplified exponentially by PCR. To cut a connection in the complementary graph, the data pool was first divided into two test tubes, t_0 and t_1. In t_0 strings containing $V_0 = 1$ were cut with Afl II, in t_1 strings containing $V_2 = 1$ were cut with Spe I. Next, t_0 and t_1 were combined into test tube t, which did not contain $***1*1$.

The maximal clique in the graph of Figure 1 is 001111 or (2, 3, 4, 5), reading from left to right. The solution is

$$P_0 \to V_0 \to P_1 \to V_1 \to P_2 \to P_3 \to P_4 \to P_5 \to P_6 \tag{1}$$

In DNA code form, the solution is represented as follows.

$$\begin{aligned} &CGTAGAATTCTGCGAACCTT \to GACGCGGCAG \\ &\to AAGAGTCACCTATCAGTAAG \to TTCATAACCG \\ &\to CTTCGTGAAGAGAGATTACT \\ &\to AGTGCTAACAAGCCTGCGCA \\ &\to TGCAGTCTTTGAGATAAAGG \\ &\to CCTACGTGTGAAATAGACTC \\ &\to GAGAGGGGCGGGATCCAGGG \end{aligned} \tag{2}$$

3 Simulated DNA Computing

We have been developing a tool for simulating DNA computing and other molecular computation methods. The tool, named NACST (nucleic acid computing simulation toolkit), is used to design the biological experiments, such as code design for solving particular problems. Ouyang's method was implemented on NACST.

In simulations we made some modifications to the original experiment which seemed more reasonable from the computational perspective. In step 1, the method of parallel overlap assembly (POA) was modified

Parameters	Values
Task	6-MCP
Pool size	1,000,000
Reaction time	1,000,000
Code length for position P_i	20
Code length for value V_i	10
Hybridization error rate	0.003
Ligation error rate	0.003

Table 2: Parameters for the experiments.

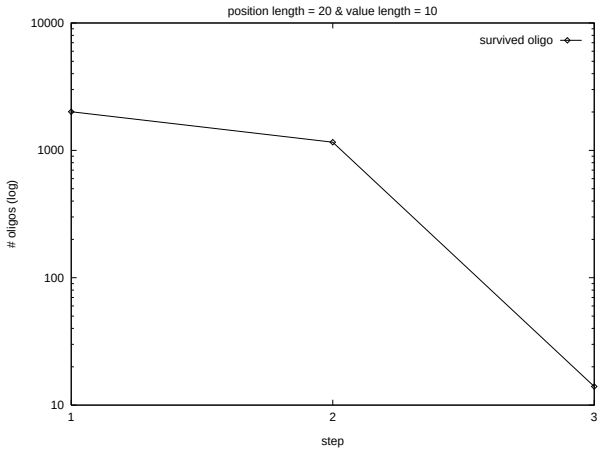

Figure 3: Number of molecules survived each lab step for simulated DNA computing.

to be compatible with the more common technique of ligation. The ligation process was emulated as realistic as biological processes. The second and third steps in the original experiment were merged into a single step. Thus in step 2 of simulation, restriction enzymes are used to cut and produce new graphs. Illegal graphs are then removed from the data pool. This process is the same as the original process. In step 3, the shortest sequence is chosen as the solution to the problem. The shortest sequence is the solution because in the presented encoding scheme the code length of V_i was set to 0 if the value of $V_i = 1$. We simulated this process by explicitly finding the sequence of 1's.

PCR plays an important role in DNA computing with biomolecules. They are used to purify the molecules to evolve useful ones. However, in simulations the process can be replaced by a more efficient search method. The use of restriction enzymes was also simulated. It was checked whether the position and value sections have the same codes as the enzymes.

Table 2 summarizes the parameters used in the simulations. The task was the 6-vertex maximal clique problem shown in Figure 1. We used the same encoding scheme as illustrated in the previous section. The code lengths for position P_i and value V_j were 20 and 10, respectively. The size of data pool was one million. The reaction time allowed was one million. Considering the huge storage capacity and parallelism of DNA molecules, this is a small amount yet sufficient to solve the problem. To simulate errors in chemical reactions we inserted point mutations during chemical reactions. Hybridization error rate was 0.3% and ligation error rate was 0.3%.

Figure 3 shows how many of the molecules survive each lab step simulated. The number of molecules survived all three steps are solutions. We studied the effect of reaction time on the reliability of computations. The results are summarized in Table 3. Given a reaction time of 10^6 the simulated DNA computing always found solutions.

Using reaction times of 5×10^5 and 2×10^5 the method also found more than one solution for each run on the average. However, given less reaction time the number of solutions found reduced. In an extreme

Position Sequence Length	Value Sequence Length	Pool Size	Reaction Time	Num Runs	Succ Runs	Num Sol's
20	10	10^6	10^6	25	25	10.36
20	10	10^6	5×10^5	25	25	6.96
20	10	10^6	2×10^5	25	22	1.76
20	10	10^6	10^5	25	12	0.68

Table 3: Comparison of the number of solutions for runs with varying reaction time.

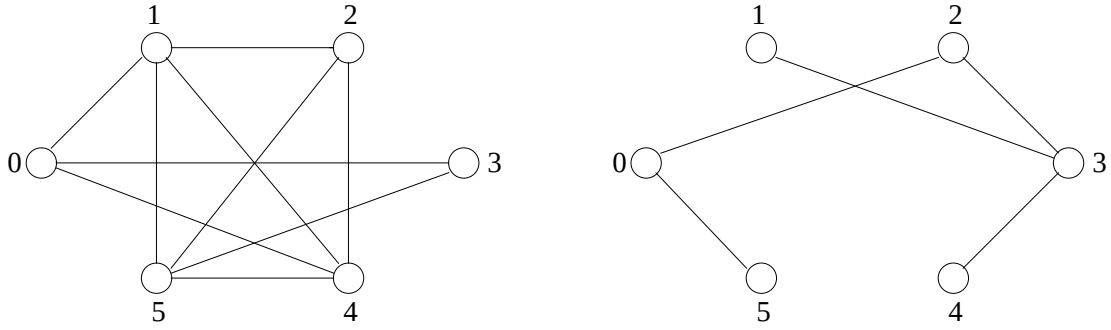

Figure 4: Another maximum clique problem: (a) graph, (b) complementary graph.

case of reaction time 10^5, the number of solutions are very small.

Simulations have also been made on a new problem instance. The graph of this new problem is depicted in Figure 4(left). Its complementary graph is shown in Figure 4(right). The solution found by simulated DNA computing is as follows.

$$\begin{aligned} &CGTAGAATTCTGCGAACCTT \rightarrow GACGCGGCAG \\ &\rightarrow AAGAGTCACCTATCAGTAAG \\ &\rightarrow CTTCGTGAAGAGAGATTACT \\ &\rightarrow AGTGCTAACAAGCCTGCGCA \rightarrow CCTGACGATT \\ &\rightarrow TGCAGTCTTTGAGATAAAGG \\ &\rightarrow CCTACGTGTGAAATAGACTC \\ &\rightarrow GAGAGGGGCGGGATCCAGGG \end{aligned} \tag{3}$$

This corresponds to the representation

$$P_0 \rightarrow V_0 \rightarrow P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow V_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_6 \tag{4}$$

which is equivalent to 011011 or $(1, 2, 4, 5)$; four vertices 1, 2, 4, and 5 construct the maximal clique for this graph.

It is interesting to note that given only 200,000 reaction steps a solution can be found. This is remarkable if we compare this result with our earlier results on Adleman-style DNA computing [9]. This is an example demonstrating that encoding schemes are very important for robustness and fidelity of DNA computing. In the following section we present yet another method that can further improve the performance of DNA computing.

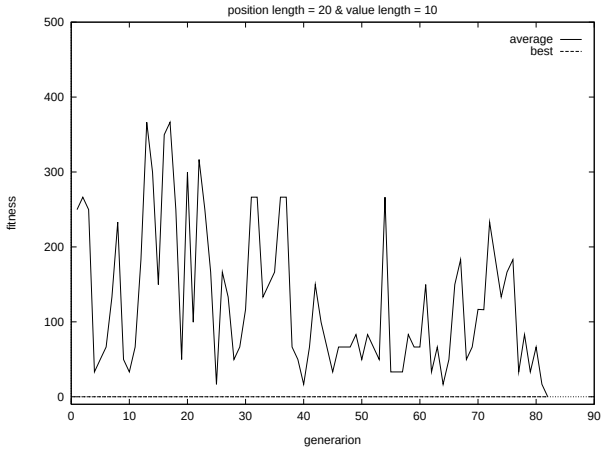

Figure 5: Fitness vs. generation for genetic code optimization.

4 Evolutionary Optimization of DNA Codes

Major errors in DNA computing described in Section 2 come from two sources. The first source of error is the production of single-stranded DNA (ssDNA) during PCR. This ssDNA cannot be cut by restriction enzymes. The second source of error is incomplete cutting by restriction enzymes, which also leads to incorrect answers.

If we carefully design the DNA codes to avoid the errors, the robustness of DNA computing can still be improved. In this section we describe a method that uses a genetic algorithm to find optimal codes for position sections P_i and value sections V_j in solving maximal clique problems.

Each encoding, i.e. a set of instantiated codes for P_i and V_i, builds an individual for the genetic algorithm. The fitness of each individual is determined by estimating the robustness of the codes when they are used in bio-lab steps. Both sources of errors are considered in determining the fitness of each code. Genetic operators are used to evolve fitter codes. Uniform crossover was used to exchange information between two codes. Point mutation was used to change the single nucleotides. The best encoding found by the genetic algorithm is then used to run DNA computing with biomolecules.

Employing the genetic optimization phase, the entire molecular algorithm can be described as follows.

1. Generate a population of encodings for the problem.
2. While (generation $g \leq g_{max}$) do
   (a) Evaluate the fitness of each encoding.
   (b) Select fitter encodings.
   (c) Apply genetic operators to produce a new population.
3. Let the best code be the fittest encoding.
4. *Initialization*: Generate a pool of oligos using the best code.
5. While (cycle $c \leq c_{max}$) do
   (a) *Synthesis*: Produce candidate solutions by molecular operators.
   (b) *Separation*: Filter out infeasible solutions by lab-steps.

The evolutionary algorithm was run with population size of 10 for 100 generations. Crossover and mutation rates were 0.5 and 0.3, respectively. Other parameters were the same as the experiments in the previous section.

Position Length	Value Length	step	Good encoding	Bad encoding
10	10	1	2650.50	2649.20
		2	1748.40	1391.00
		3	3.60	0.25
20	10	1	2648.90	2648.90
		2	1433.50	759.00
		3	3.25	0.00
30	10	1	2653.85	2654.21
		2	1448.95	1258.37
		3	3.90	0.53

Table 4: Number of molecules survived the lab steps: comparison with good (optimized) and bad (negatively optimized) encodings using position length $|P_i| = 10, 20, 30$ for reaction time = 1000000.

Figure 5 depicts the fitness evolution during the genetic code optimization process. It can be seen that encodings that can solve the problem are found in early generations. We also studied the effect of the code length on the fidelity of DNA computing. Experiments were performed with three varying values of position code length: 10, 20, 30.

Table 5 shows a good (optimized) encoding. To see the difference, we also ran the genetic algorithm to find bad encodings by negatively optimizing the codes. Table 6 shows an example of bad (negatively optimized) codes found by the genetic search. One interesting feature of bad codes is that they have the same sequence as a restriction enzyme or sequences that can be the same if mutated. This leads to cutting wrong positions.

Table 4 compares the number of molecules that survived each simulation lab-step. This is a measure of reliability of DNA computing. Both of the good and bad encodings well survived the first two steps, but DNA computing with good encodings resulted in a larger number of solutions than with bad encodings. The reason for no big difference between both encodings seems attributed to the fact that the presented encoding scheme gives so much constraints on the possibility of mismatches that there are no much room for finding better codes. This is contrasted with our own results on Adleman-style DNA encoding reported in [9], where the genetic optimization of codes have got more significant improvement in real experiments.

Significant errors are caused when a sequence has one or more subsequences that are the same as the codes for restriction enzymes. This leads to wrong cutting of the sequences. This type of error is affected by the length of position sequences. In case of position length 20, there are 10 to 20% of the cases where unnecessary restriction enzymes are placed within the position sequences. Even with mutations, the rate of wrong positions ranges between 20 and 30%. This suggests a possibility of error tolerance of the encoding scheme.

If we shorten the length of position sequences, say 10, then there are only four positions that may cause mismatches (remaining six are for two restriction enzymes for both ends of the sequence). This should improve the fidelity of DNA computing. The simulation results for $|P_i| = 10, 20, 30$ conform to this analysis. The result shows the effect that restriction enzymes place wrong positions in the sequence. Thus care must be paid in interpreting the results in Table 4.

In contrast to position sequences, restriction enzymes on value sequences have little effect on solutions. To understand why, recall that $V_i = 1$ if a value sequence exists and $V_i = 0$ otherwise. Though a value sequence has the same code as the restriction enzyme, if the solution does not contain the value sequence, then this has no effect on the solution at all. The only case affecting the result is the case that the value sequence has the same code as the restriction enzyme and this value sequence is required for the solution of the problem. For example, Table 6 includes restriction enzyme "CTTAAG" for the value sequence V_0 which is a part of the answer sequence. Hybridization errors have little effect since the restriction enzymes play the primary role in distinguishing sequences.

Length		DNA	Good Encoding
Position	Value	fragment	
		$P_0V_0^0P_1$	CTCGTACAAATTGGGTTCTTgcggccttgc
			AAGCTGGTACCCGGACCAAG
		$\underline{P_0V_0^1P_1}$	CTCGTACAAATTGGGTTCTTAAGCTGGTACCCGGACCAAG
		$\overline{P_2V_1^0P_1}$	AGTTGTACCTGCTCCCAAAGgccagatcgt
			CTTGGTCCGGGTACCAGCTT
		$\overline{P_2V_1^1P_1}$	AGTTGTACCTGCTCCCAAAGCTTGGTCCGGGTACCAGCTT
		$P_2V_2^0P_3$	CTTTGGGAGCAGGTACAACTtcgccccata
			AGTAGACCTACGCGAGAGCA
20	10	$\underline{P_2V_2^1P_3}$	CTTTGGGAGCAGGTACAACTAGTAGACCTACGCGAGAGCA
		$\overline{P_4V_3^0P_3}$	CCTCTGCCGCGAGTAATGCAatgaatgttt
			TGCTCTCGCGTAGGTCTACT
		$\overline{P_4V_3^1P_3}$	CCTCTGCCGCGAGTAATGCATGCTCTCGCGTAGGTCTACT
		$P_4V_4^0P_5$	TGCATTACTCGCGGCAGAGGtcttgaggat
			CCTGGCAAGTCATTCGTCTC
		$\underline{P_4V_4^1P_5}$	TGCATTACTCGCGGCAGAGGCCTGGCAAGTCATTCGTCTC
		$\overline{P_6V_5^0P_5}$	TACCAGGTCCGCGGACGCTCagagtaagcg
			GAGACGAATGACTTGCCAGG
		$\overline{P_6V_5^1P_5}$	TACCAGGTCCGCGGACGCTCGAGACGAATGACTTGCCAGG

Table 5: Good codes found by the code optimization process for $|P_i| = 20$.

Length		DNA	Bad Encoding
Position	Value	fragment	
		$P_0V_0^0P_1$	CTCGAGTCAAGCACTGGCTTaacttaagtc
			AAGCACACGAAGCCCGCAAG
		$\underline{P_0V_0^1P_1}$	CTCGAGTCAAGCACTGGCTTAAGCACACGAAGCCCGCAAG
		$\overline{P_2V_1^0P_1}$	AGTGAATTAATTCCCGAAAGagtagccaat
			CTTGCGGGCTTCGTGTGCTT
		$\overline{P_2V_1^1P_1}$	AGTGAATTAATTCCCGAAAGCTTGCGGGCTTCGTGTGCTT
		$P_2V_2^0P_3$	CTTTCGGGAATTAATTCACTtgaccccctg
			AGTATACCTAATTAATGGCA
20	10	$\underline{P_2V_2^1P_3}$	CTTTCGGGAATTAATTCACTAGTATACCTAATTAATGGCA
		$\overline{P_4V_3^0P_3}$	CCTCGTGGCTAACTGTTGCAttgtgtccac
			TGCCATTAATTAGGTATACT
		$\overline{P_4V_3^1P_3}$	CCTCGTGGCTAACTGTTGCATGCCATTAATTAGGTATACT
		$P_4V_4^0P_5$	TGCAACAGTTAGCCACGAGGcttcgtatgc
			CCTAATACTAGTACGGCCTC
		$\underline{P_4V_4^1P_5}$	TGCAACAGTTAGCCACGAGGCCTAATACTAGTACGGCCTC
		$\overline{P_6V_5^0P_5}$	TCTACAACCCTCGCCCCCTCaagcaactaa
			GAGGCCGTACTAGTATTAGG
		$\overline{P_6V_5^1P_5}$	TCTACAACCCTCGCCCCCTCGAGGCCGTACTAGTATTAGG

Table 6: Bad codes found by the code optimization process for $|P_i| = 20$.

5 Conclusions

We presented an evolutionary method for the design of DNA codes in DNA computing. In the context of the maximal clique problem we demonstrated the effectiveness of the optimized codes. The general observation from our simulation results is that the encoding scheme used by Ouyang et al. are very robust against errors. This is contrasted with our earlier simulations of Adleman-style DNA computing. The main strength of the new encoding scheme lies in the use of restriction enzymes which rapidly cut out useless code fragments. Though the effect is less significant, using the genetic algorithm to optimize the DNA codes for the new encoding scheme still improves the performance of DNA computing.

Acknowledgments

This research was partially supported by grants from the Korea Science and Engineering Foundation (KOSEF 981-0920-107-2), the Dongbu Foundation, and the IBM Korea.

References

[1] L.M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, 266:1021-1024, 1994.

[2] R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, S.E. Stevens, Jr., Good encodings for DNA-based solutions to combinatorial problems, In *Proc. of Second DIMACS Workshop on DNA Based Computers*, American Mathematical Society, 1996.

[3] M.R. Garey and D. S. Johnson, *Computers and Intractability*, New York: Freeman, 1979.

[4] M. Garzon, P. Neathery, R. Deaton, R. C. Murphy, D.R. Franceschetti, and S.E. Stevens Jr., A new metric for DNA computing, *Proc. Second Annual Conf. on Genetic Programming*, Koza et al. (eds.), MIT Press, pp. 472-478, 1997.

[5] T. Haynes and A. Wu, *ICGA-97 Workshop on Exploring Non-coding Segments and Genetics-based Encodings*, East Lansing, Michigan, 1997.

[6] R.J. Lipton, DNA solution of hard computational problems, *Science*, 268:542-545, 1995.

[7] Q. Ouyang, P.D. Kaplan, S. Liu, and A. Libchaber, DNA solution of the maximal clique problem, *Science*, 278:446-449, 1997.

[8] B.T. Zhang and H. Mühlenbein, Balancing accuracy and parsimony in genetic programming, *Evolutionary Computation*, 3(1):17–38, 1995.

[9] B.T. Zhang and S.Y. Shin, Molecular algorithms for efficient and robust DNA computing, *Genetic Programming 1998*, Koza, J. R. *et al.* (eds.), Morgan Kaufmann, pp. 735-742, 1998