Molecular Algorithms for Efficient and Reliable DNA Computing

Byoung-Tak Zhang
Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering
Seoul National University
Seoul 151-742, Korea
Phone: +82-2-880-1833
btzhang@scai.snu.ac.kr

Soo-Yong Shin
Artificial Intelligence Lab (SCAI)
Dept. of Computer Engineering
Seoul National University
Seoul 151-742, Korea
Phone: +82-2-880-1835
syshin@scai.snu.ac.kr

ABSTRACT

Two new molecular algorithms are presented that are designed for the improvement of efficiency and reliability of DNA computing. The first algorithm introduces an evolutionary cycle to guide chemical reactions of DNA molecules. The second molecular algorithm extends the first one by adding another evolutionary loop for optimizing encodings of the problem instance. Just as genetic programming is a method for programming conventional computers by means of natural evolution, our approach, which might be called *molecular programming*, provides a method for programming biocomputers by means of artificial evolution. Simulations have been performed with the Hamiltonian path problem to verify the positive effect of the presented molecular algorithms on the reliability and efficiency of DNA computing.

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 [9] came up with using DNA computing to solve the satisfiability (SAT) problem: given a propositional formula decide if it is satisfiable. He showed how to use the same primitive DNA operations as Adleman to solve any SAT problem with N binary inputs and G logical gates (AND, OR, or NOT), in a number of operations depending linearly on $N + G$.

In DNA computing two parameters are considered. One is the time τ required for separation of molecules. The other is the accuracy, i.e. the error rate of a separation. In a real system separation will not be perfect, there will be errors of false positives and false negatives. It is also possible that in real systems the time and accuracy of a separation may depend on the tube and encodings being considered.

From the computer science point of view there are several points of improvements in the Adleman and Lipton constructions. Since ordinary sequential computers are not restricted to using naive brute force algorithms to solve NP-complete problems such as HPP and SAT. Instead, they can use clever algorithms, which, while still exponential, tend in practice to have much smaller growth constants than the naive algorithm.

Another point is the encoding of problem instances for DNA computing. 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. Deaton et al. [3, 5] have identified various types of errors that lead to false positives in Adleman's original techniques. They also gave a theoretical bound on the size of problems that can be solved reliably and used genetic search for good encodings. Garzon et al. [6] introduced a new measure of hybridization likelihood and proposed a theory of error-preventing codes for DNA computing.

Motivated by the work of Deaton and Garzon we investigate in this paper methods for further improvement of the efficiency and robustness of DNA computing. We introduce two variants of molecular algorithms used in the Adleman's original experiment. The algorithms are inspired by genetic programming. Just as genetic programming is a method for programming "digital" com-

puters by means of "natural" evolution, our approach, which might be called *molecular programming*, provides a method for programming "biocomputers" by means of "artificial" evolution. Genetic programming is based on tree-structured representations while molecular programming uses double strands of DNA sequences. Just as trees are natural representations for conventional digital computers, DNA molecules are natural representations for biocomputers. The operators in genetic programming are inspired by natural systems, whereas the operators in molecular programming are inspired by artificial systems.

The first molecular programming algorithm we studied is an "evolutionary" version of simple DNA computing. Instead of brute force search, the algorithm "evolves" molecules by iterating synthesis and separation steps. The second molecular algorithm is an extended version of the first algorithm that has another evolutionary loop outside the molecular evolution cycle. This outer genetic search optimizes DNA encodings for representing problem instances. It should be noted that our approach is different from that of Deaton at al. [4]. While they focuse on implementing a genetic algorithm by making use of mismatches in chemical reactions, we attempt to predict and minimize such mismatches to make DNA computing more efficient and reliable.

The effectiveness of the proposed algorithms is shown on the Adleman's graph for the 7-city Hamiltonian path problem. Our simulation results using the NACST (Nucleic Acid Computing Simulation Toolbox) system which are being developed in the SCAI Lab at Seoul National University indicate that the Adleman's encoding scheme with random 20-mers are effective from biological point of view, but are inefficient from the computing point of view. Our simulations suggest that there exist shorter encodings that can effectively solve the 7-HPP problem.

The paper is organized as follows. In Section 2 we describe the basic molecular algorithm for the Adleman-style DNA computing. Sections 3 and 4 present two new molecular algorithms and their experimental results on the improvement of reliability and efficiency of DNA computing. Section 5 summarizes our findings from this study and suggests further work.

2 Adleman-Style DNA Computing

Adleman [1] showed how to use molecules of DNA to solve an instance of the Hamiltonian path problem in a test tube using standard methods of molecular biology. Given a set of 7 vertices with some interconnecting (directed) paths, the problem is to find a path which visits each vertex exactly once. Figure 1 shows a graph that

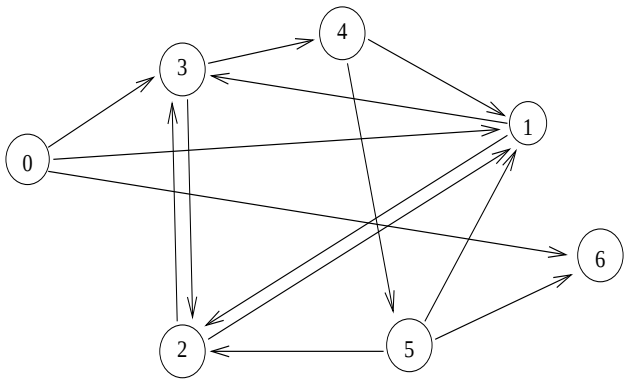


Figure 1: **The Hamiltonian path problem with 7 vertices.**

has a Hamiltonian path from $v_{in} = 0$ to $v_{out} = 6$, given by the edges

$$0 \to 1,\ 1 \to 2,\ 2 \to 3,\ 3 \to 4,\ 4 \to 5,\ 5 \to 6. \qquad (1)$$

Adleman synthesized unique 20-mer nucleotide tag sequences to give each vertex a "from" and "to" address. For each vertex i in the graph, a random 20-mer oligonucleotide

$$O_i \qquad (2)$$

is generated. For edge $i \to j$ in the graph, an oligonucleotide

$$O_{i \to j} \qquad (3)$$

is derived from the 3' 10-mer of O_i and from the 5' 10-mer of O_j. For each vertex i in the graph,

$$\bar{O}_i \qquad (4)$$

is the Watson-Crick complement of O_i. For example, $\bar{O}_5$ serves as a splint to bind $O_{4\to5}$ and $O_{5\to6}$ in preparation for ligation.

The following nondeterministic algorithm solves the problem:

1. Generate random paths through the graph.
2. Keep only those paths that begin with v_{in} and end with v_{out}.
3. If the graph has n vertices, then keep only those paths that enter exactly n vertices.
4. Keep only those paths that enter all of the vertices of the graph at least once.
5. If any paths remain, say "Yes"; otherwise, say "No."

Each path was represented by the 140-mer containing the appropriate path tags in the correct order. In solution these single-stranded DNA molecules randomly hybridized to their complements, forming longer strands which were double-stranded in complementary regions. After allowing about four hours for the strands to hybridize, DNA ligase and polymerase were used to "sew up" the strands, getting a set of DNA molecules representing all possible paths in the graph. Chemical and physical operations were then performed to extract only the DNA molecules which corresponded to Hamiltonian paths from this soup.

The total number of separation and synthesis steps required grows linearly with the size (number of vertices and edges) of the problem. The experiment took Adleman about a week.

The essential steps needed for DNA computing above can be summarized as follows:

1. *Encoding*: Determine an encoding for the problem.
2. *Initialization*: Generate a pool of oligos.
3. *Synthesis*: Produce candidate solutions by molecular operators.
4. *Separation*: Filter out infeasible solutions by lab steps.

In the Adlemans experiment the encoding for oligos were random and there were no control over the synthesis and separation steps. One drawback of this simple molecular algorithm is that it might propagate only bad solutions in the synthesis step. For example, the candidates longer than 140-mer are infeasible solutions but may continue to hybridize to generate other infeasible solutions. Useless reactions can be avoided by early application of separation steps. The following molecular algorithm is based on this observation.

3 A Molecular Evolutionary Algorithm

We introduce an evolutionary loop into the basic molecular algorithm for DNA computing. In this algorithm, the synthesis and separation steps are respectively equivalent to genetic operators and selection in genetic programming. The molecular evolutionary process can be summarized as follows.

1. *Encoding*: Determine an encoding for the problem.
2. *Initialization*: Generate a pool of oligos.
3. 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 procedure incorporates a while loop that iterates the synthesis and separation steps for c_{max} cycles. This molecular algorithm is more like an evolutionary algorithm in the sense that the DNA pool evolves as time goes on. Unlike usual evolutionary algorithms, the selection operators are based on binary constraints. If the constraint is satisfied, then DNA sequences kept in the test tube, otherwise they are filtered out. There is no fitness function defined explicitly.

We compared the performance of the simple molecular algorithm (sMA) and the iterative version (iMA). Simulations have been performed on the NACST system for the 7-vertex HPP problem. The performance was measured in terms of efficiency and reliability. In simulations, we use two parameters to control the bio lap experiment. One is the pool size, defined as the number of oligos, to determine the cost for biomaterials. The other parameter is the reaction time, the maximum number of reactions allowed in the synthesis step, which determines the time for DNA computing in simulation.

The sMA algorithm was initialized with 1 million oligos which undergo chemical reactions for 10 million times. The iMA was given the same number of oligos but it iterates 10 cycles of synthesis and separation steps. One million (one tenth of the sMA) reactions take place for each synthesis step.

In the iMA the molecules were removed from the pool when its code starts with v_{in} and ends with v_{out} but the length is shorter than the solution. Pruning was also performed in the case that the sequence is longer than the solution length. By removing infeasible molecules and then continuing reactions, the iterative algorithm prevents consuming oligos without producing feasible solutions. This results in more effective use of resources (oligos) and thus reduce the costs for lab experiments.

We studied the effect of reaction time on the reliability of finding solutions. The results are summarized in Table 3. For various reaction times, the iterative molecular algorithm found solutions approximately twice more often than the simple molecular algorithm. The simple molecular algorithm found solutions reliably only under enough reaction time and a big size of pool.

Figure 2 compares the number of molecules remaining in the pool as the experiment step progresses. It can be observed that, whereas the size of oligos for sMA is relatively large in step 2 and undergoes a fast reduction in step 3, the size of oligos for iMA reduces by a large amount at step 2 and remains almost the same thereafter. The result is that iMA produces a larger

Table 1: **Parameters for the simple molecular algorithm.**

Parameters	Values
Task	7-HPP
Pool size	1,000,000
Reaction time	10,000,000
Max cycle	1
Code length for vertex	10
Hybridization error rate	0.01
Ligation error rate	0.01

Table 2: **Parameters for the iterative molecular algorithm.**

Parameters	Values
Task	7-HPP
Pool size	1,000,000
Reaction time	1,000,000
Max cycle	10
Code length for vertex	10
Hybridization error rate	0.01
Ligation error rate	0.01

Table 3: **Comparison of the simple molecular algorithm (sMA) and the iterative molecular algorithm (iMA) in the average number of solutions.**

Code Length	Pool Size	Reaction Time	Num Runs	Avg# Solutions sMA	iMA
20	10^6	10^7	20	1.70	3.10
20	10^6	7.5×10^6	20	1.40	2.20
20	10^6	5×10^6	20	1.10	1.80

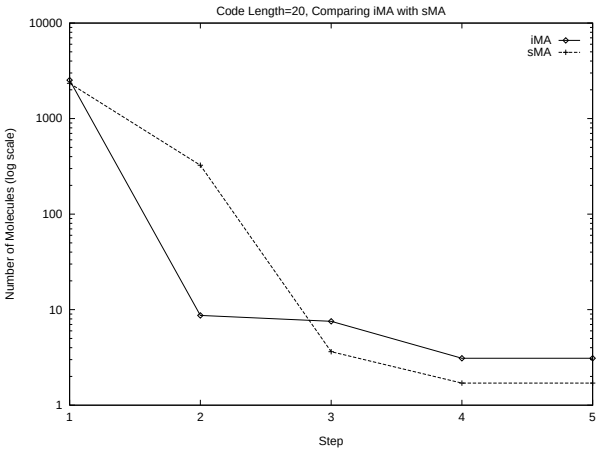


Figure 2: **Number of molecules survived each lab step for the simple molecular algorithm.**

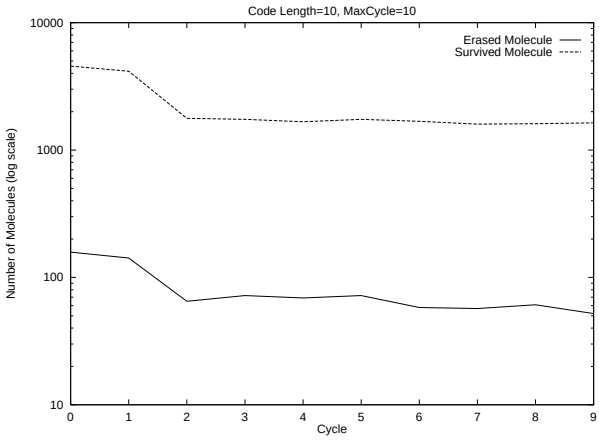


Figure 3: **Number of molecules survived each synthesis-separation cycle for the iterative molecular algorithm.**

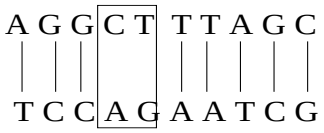
number of solutions with faster convergence than sMA. This demonstrates the effect of iMA to remove infeasible solutions at an early stage.

Figure 3 shows the survived and erased molecules in each iteration. The amount of survived molecules is large at cycles 0 and 1, after which it remains almost constant. Similar phenomenon is observed for the erased molecules.

4 A Molecular Evolutionary Algorithm with Code Optimization

The main feature of the code-optimizing molecular algorithm is to search for good encodings before starting the bio experiments. Deaton et al. [3] has explored the use of a genetic algorithm for the optimization of the encoding. This concept is combined with the iterative molecular algorithm, resulting in a molecular evolutionary algorithm with code-optimization:

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) Mismatched Hybridization

(b) Shifted Hybridization

Figure 4: **Types of reaction error.**

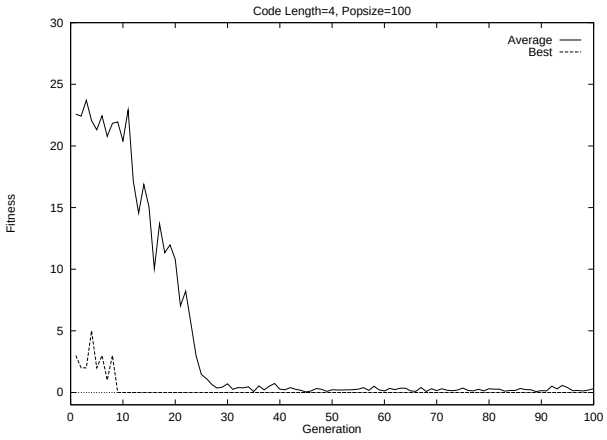


Figure 5: **Fitness vs. generation for genetic code optimization.**

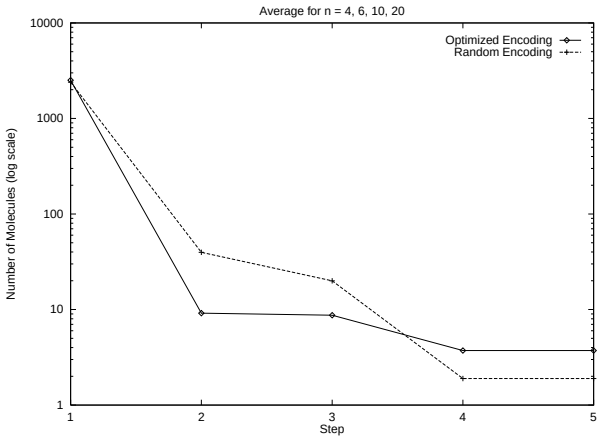


Figure 6: **Number of molecules survived each lab steps: comparison of the molecular algorithm with and without code optimization.**

(a) *Synthesis*: Produce candidate solutions by molecular operators.

(b) *Separation*: Filter out infeasible solutions lab-steps.

In effect, the encoding step of the iterative molecular algorithm is extended by steps 1–3 above to generate an optimized encoding.

To find good encodings we need to consider various causes of errors. There are two main types of errors in DNA computing. False positives are the error caused when the algorithm produces sequences which appear to be valid solutions, but actually are not. False negatives are the answers to the problem which exist, but the algorithm cannot find. These errors are caused by mismatches in chemical reactions. We currently consider the following types of reaction errors as illustrated in Figure 4.

1. Mismatched hybridization
2. Shifted hybridization

Hybridization stringency refers to the number of complement base pairs that have to match for DNA oligonucleotides to bind. Hybridization stringency depends on reaction conditions, such as sodium concentration, temperature, and relative percentage of A+T's and G+C's. The indirect effect of these factors on hybridization errors are not considered in the following simulations.

For solving the HPP problem, we considered three types of errors: false positive hybridiztion of verticies, false negative hybridization of vertices, and correct hybridization using illegal edges. Thus, the fitness of an encoding are measured by estimating these possibilities.

Table 4 describes the experimental setup for DNA computing simulations. Experiments were performed with four varying values of code length: 4, 6, 10, 20. Other parameters are the same as the experiments in the previous section. Table 5 shows the parameters used for genetic optimization of encodings for the 7-HPP problem. An evolutionary algorithm was run with population size of 100 for 100 generations. Crossover rate and mutation rate were 0.5 and 0.1 respectively.

Figure 5 depicts the fitness evolution during the genetic code optimization process. It can be seen that fit encodings are found within 10 generations. The reliability of the encoding was verified by running the DNA computing simulator. The result of optimized encoding and random encoding are compared in Figure 6. The graph shows the number of molecules survived each lab step, averaged over four diffent code lengths (n = 4, 6, 10, 20). Optimized encodings produced less molecules in steps 2 and 3, but more molecules in steps 4 and 5, thus more solutions, than random encodings.

Table 8 shows examples of good (optimized) and bad (negatively optimized) encodings found by the genetic

Table 4: **Parameters for the iterative molecular algorithm.**

Parameters	Value
Task	7-HPP
Pool size	1,000,000
Reaction time	1,000,000
Max cycle	10
Code length for vertex	4, 6, 10, 20
Hybridization error rate	0.01
Ligation error rate	0.01

Table 5: **Parameters for genetic codeword optimization.**

Parameter	Value
Task	7-HPP
Population size	100
Max generation	100
Crossover rate	0.5
Mutation rate	0.1

Table 6: **Number of molecules survived the lab steps: comparison with good (optimized) and bad (negatively optimized) encodings for** $n = 4, 6, 10, 20$**.**

n	step	Good encoding	Bad encoding
4	1	2504.59	2436.29
	2	8.93	0.95
	3	8.44	0.33
	4	3.81	0.00
	5	3.81	0.00
6	1	2506.60	2468.10
	2	9.04	59.57
	3	8.72	56.86
	4	3.92	4.38
	5	3.92	4.38
10	1	2515.00	2161.15
	2	9.95	17.55
	3	8.95	16.75
	4	4.05	2.70
	5	4.05	2.70
20	1	2526.60	2534.95
	2	8.70	2.73
	3	7.55	1.95
	4	3.10	0.50
	5	3.10	0.50

Table 7: **Comparison of the molecular algorithm with random codes and optimized codes in the number of successful runs (pool size=10^6, reaction time=10^7).**

Code Length	Num Runs	#Successful Runs Optimized	Random
4	20	20	0
6	20	20	20
10	20	20	19
20	20	20	9

search. Listed are the encodings for code length $n = 4, 6, 10, 20$.

Table 6 compares the number of molecules that survived each lab step. As can be seen, molecules of bad encodings have in general a smaller number of solutions than optimized encodings. An exception is the case of $n = 6$ in which the random encoding has a slightly larger number of solutions (3.92 vs. 4.38), but the andom encoding used about 7 times more molecules than that of the optimized encoding. This demonstrates the possibility of the iterative algorithm for improving the efficiency of compuation and usage of molecules.

5 Conclusions

We presented two DNA computing algorithms that are inspired by evolutionary algorithms. The introduction of inner evolutionary cycle increased the efficiency of the DNA computing procedure. The use of outer evolutionary loop reduced the error rate in hybridization process and thus improved reliability of DNA computing. Our molecular algorithms are motivated more from the computer science point of view rather than the biological point of view, and so far we have ignored some of biolab techniques which are also dependent of temperature, sodium concentration etc.

It should be noted that the utility of the augmented molecular algorithms in real molecular systems depends on the types of lab techniques adopted. The simpler the separation steps, the more useful the inner evolutionary loop. On the other hand, if separation steps are more time-consuming than hybridization, then the iteration may be more expensive in time. However, in this latter case there still exists the possibility of utility; it makes better use of molecules since the molecular evolutionary algorithm makes better use of the molecules than the brute force DNA computing procedure.

We identified similarities between DNA computing and genetic programming. Both are similar in that the

Table 8: **Good and bad encodings found by the code-optimizing molecular algorithm:** $n = 4, 6, 10, 20$.

n	Vertex	Good Encoding	Bad Encoding
4	0	GATT	AGTA
	1	CAGC	AGAA
	2	TACA	CAGT
	3	AATA	AGCC
	4	GCTC	ACTG
	5	GTAC	TATA
	6	TCCC	CTTG
6	0	TCGTGA	TACTAG
	1	CGAATT	TCCGAA
	2	GCCAAA	ATTCGG
	3	CAATTC	GCTAAG
	4	GTCTTG	AAGAAT
	5	GGTAAG	ATTACA
	6	GGACGC	TCCGAC
10	0	ACGTAGCTCC	GGTTCCGTGA
	1	TACGTCGATC	ATATGCAGGC
	2	GTACCAATGC	AAAACTGTAG
	3	CGTAGTCATG	CGGTTCTCCC
	4	ATGCCTTCAG	CGGTTGGGAA
	5	AGCGTATCGT	ACGCTCCCAG
	6	CAATACTTCT	TATGACGTAT
20	0	AAAAAACATC GGAATACATG	ACATGGATCA TGCAGTGTCT
	1	CAGCATAAAC CTGTGGAAGA	CGCCTTTCGT AGAACTGCCG
	2	ATGTCGGAGT TCACCTACCA	CCGTGGGCTG GATTTTCGGA
	3	ACCACATGCG GCGGGCGACA	ATCGTAAGTG GCTACTAGCA
	4	GCTCACTGCT TCGAAAGCTC	TTTAACAATG AATTGAACAA
	5	GGTATCATCG TTTGGACTGG	TTTAGCGTTT ACAGGTGTTT
	6	CTTTATATCA GGAATAAGTG	GACTATTTGT TGCAGTGTCT

size of encoding is variable. The difference lies in that GP has tree structures while DNA molecules has linear structures (at least in current encodings). Many genetic programming researchers have faced with the problem of bloating, the explosion of non-functional codes [7]. We also have observed similar non-functional codes in DNA computing. The presented molecular algorithms were motivated from this observation and used to reduce the bloating factor in DNA computing.

In this paper we confined ourselves to Adleman-style DNA computing applied to the Hamiltonian path problem. It would be more interesting to see our molecular algorithms solve other computational problems.

Acknowledgments

Thanks to Sung-Soo Jun (Department of Molecular Biology), Gye-Won Seo (Department of Chemistry), Sirk-June Augh (Institute for Molecular Biology and Genetics), all with Seoul National University, for helpful discussions. This research was partially supported by the Dongbu Foundation and the Seoul National University Research Foundation.

References

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

[2] Adleman, Leonard M. 1996. On constructing a molecular computer. *DNA Based Computers*, American Mathematical Society. Pages 1-21.

[3] Deaton, R., Murphy, R.C., Garzen, M., Franceschetti, Donald R., Stevens, S.E. Jr.. 1996. Good encodings for DNA-based solutions to combinatorial problems, *Proceedings of 1996 Second DIMACS Workshop on DNA Based Computers*, American Mathematical Society, (to appear)

[4] Deaton, R., Murphy, R.C., Garzon, M., Franceschetti, Donald R., Stevens, S.E. Jr.. 1997. A DNA based implementation of an evolutionary search for good encodings for DNA computation. *Proceedings IEEE Conference on Evolutionary Computation*, IEEE Press. Pages 267-271.

[5] Deaton, R., Murphy, R.C., Garzen, M., Franceschetti, Donald R., Stevens, S.E. Jr.. 1998 Reliability and efficiency of a DNA-based computation. *Physical Review Letters*, 80(2) 417-420.

[6] Garzon, M., Neathery, P., Deaton, R., Murphy, R. C., Franceschetti, D.R., and Stevens, S.E. Jr..

1997. A new metric for DNA computing. Koza, John R., Deb, K., Dorigo, M., Fegel, David B., Garzon, M., Iba, H., and Riolo, Rick L., (editors). *Genetic Programming 1997: Proceedings the Second Annual Conference*, The MIT Press. Pages 472-478.

[7] Haynes, T., and Wu, A. 1997. *International Conference on Genetic Algorithms-97 Workshop on Exploring Non-coding Segments and Genetics-based Encodings*, East Lansing, Michigan.

[8] Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA: The MIT Press.

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

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

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

[12] Zhang, B.T. and Shin, S.Y. 1998. *Code Optimization for DNA Computing of Maximal Cliques*, Seoul National University Computer Engineering Department Artificial Intelligence Lab (SCAI) technical report SCAI-98-008. March, 1998.