Solving Traveling Salesman Problems Using Molecular Programming

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

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

Sung-Soo Jun
Dept. of Biology
Seoul National University
Seoul 151-742, Korea.
ssjun@scai.snu.ac.kr

Abstract- Molecular programming (MP) has been proposed as an evolutionary computation algorithm at the molecular level [Zhang and Shin, 1998]. MP are different from other evolutionary algorithms in its representation of solutions using DNA molecular structures and its use of bio-lab techniques for recombination of partial solutions. In this paper, molecular programming is applied to traveling salesman problems (TSPs) whose solution requires encoding of real-values in DNA strands. We propose a new encoding scheme for real values that is biologically plausible and has a fixed code length. The effectiveness of the proposed method is verified by simulations and by comparison with Narayanan and Zorbalas'.

1 Introduction

The field of DNA computing was pioneered by Adleman [1] who showed the potential of using biomolecules for solving computational problems. He solved the hamiltonian path problem using DNA molecules, and Lipton came up with a method using DNA computing to solve the satisfiability (SAT) problem [6]. While the conventional computer programs express problems in bits and solve them using the electronic computer, DNA computing represents the problems with DNA molecules and solves them by biological laboratory techniques. DNAs consist of four types of nucleotides: A (Adenine), T (Thymine), G (Guanine), and C (Cytosine). Biological techniques include hybridization, ligation, polymerase chain reaction (PCR), gel electrophoresis, and affinity column and the other techniques.

One advantage of DNA computing is massive parallelism. Another benefit is its enormous information storage capacity. 1μ mol of DNA contains approximately 10^{20} molecules which allow a big problem space can be searched in almost constant time.

Though very promising, DNA computing technology of today has some difficulties [8]. One of them is the accuracy problem; False positives and false negatives take place. Current biological laboratory techniques have the possibilities of errors such as hybridization mismatches and extraction errors. Another problem is the large cost for biological lab experiments. Both problems can be alleviated by a software tool which simulates DNA computing in advance to optimize DNA codes and experimental conditions.

Molecular programming (MP) has been proposed for these purposes. MP is a tool for programming DNA computers by means of artificial evolution [12]. Just as conventional evolutionary algorithms are a method for programming "digital" computers by means of "natural" evolution, MP provides a method for programming "biocomputers" by means of "artificial" evolution. On the other hand, MP can be regarded as a new evolutionary computation method that represents problems in DNA double strands and uses bio-lab techniques as search operators. NACST (Nucleic Acid Computing Simulation Toolkit) is an implemented MP system [12].

In this paper, we use molecular programming to solve traveling salesman problems (TSPs). Most of the problems solved so far by DNA computing have binary-valued edges. This is also true for the hamiltonian path problem [1, 12] and the maximal clique problem [10, 13]. In contrast, TSPs have costs (usually real-valued) associated with edges. Recently, Narayanan and Zorbalas [9] proposed a method to represent weights in DNA codes to solve traveling salesman problems. However, this method uses a code length which is proportional to the absolute value of the weights and thus its code length grows fast as the number of resolution of weights increases. In addition, this method seems not appropriate for representing real values.

We present an encoding scheme that uses fixed-length codes for representing integer and real values. This method is based on the fact that hybridization between G/C pairs occur more frequently that those between A/T pairs. This is because there are 3 hydrogen bonds between G and C, whereas 2 hydrogen bonds between A and T. The robustness of the method is verified by simulations on TSPs and compared with other methods.

The paper is organized as follows. In Section 2, we present the coding scheme for solving the traveling salesman problems. Section 3 describes the molecular programming method: the genetic algorithm for the optimization of the DNA codes and the molecular algorithm for efficient evolution of DNA solutions. Sections 4 and 5 provide the experiment results and conclusions.

0-7803-5536-9/99/$10.00

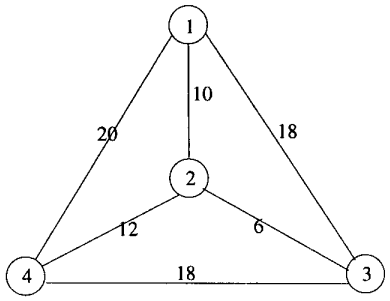

Figure 1: A graph for the traveling salesman problem.

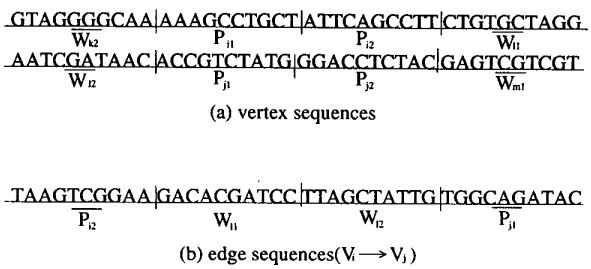

Figure 2: The fixed length coding scheme.

2 DNA Coding for TSP Solutions

2.1 Traveling Salesman Problem

The traveling salesman problem is to find a minimum cost path for a given set of vertices (cities) and edges (roads). In addition, the solution path must contain all the cities given, each only once, and begin from the specified city to which the tour ends. More formally [4], given a set C of m cities, distance $d(c_i, c_j) \in Z^+$ for each pair of cities $c_i, c_j \in C$, and a positive integer B, the TSP is to determine a tour of C having length B or less, i.e., a permutation $\langle c_{\pi(1)}, c_{\pi(2)}, \ldots, c_{\pi(m)} \rangle$ of C such that

$$\left(\sum_{i=1}^{m-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \right) + d(c_{\pi(m)}, c_{\pi(1)}) \leq B. \qquad (1)$$

Figure 1 shows an instance of a graph that has 4 nodes. Let the starting vertex is 1. Then, the path with minimum cost be $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$ and the cost of this path is 54.

2.2 Coding Scheme

Most existing DNA computing methods follow the Adleman's coding scheme to solve NP-complete problems. Here, the vertex codes are generated at random and then the edge codes, which link the vertices, are produced using the vertex codes. Ouyang et al. [10] proposed a new coding scheme in which the edges have binary values coded in sequences.

Recently, Narayanan and Zorbalas proposed DNA algorithms to solve traveling salesman problems [9]. They describes two methods for representing edges in graphs. The first method is as follows:

1. For the binary path $X \rightarrow Y$, join n occurrences of the DNA sequence for X with one occurrence of the DNA sequence for Y.
2. For the binary path $Y \rightarrow X$, join n occurrences of the DNA sequence for Y with one occurrence of the DNA sequence for X.

This method duplicates the vertex sequences to represent the edge weights. The second method is an improved version of the first.

1. Create a-variable length sequence S_i.
2. The length l_i of S_i is proportional to the weight w_i of edge i, where the length is increased by a constant factor k: $l_i = w_i \cdot k$.

This method introduces the weight sequences proportional to the edge weights in edge sequences. For example, if the edge weights are $\{2, 4\}$ and the constant factor k is 2, the lengths of weight sequences are $\{4, 8\}$, so the total length of edge sequences is varied by weight sequences. Basically these coding schemes take advantage of the DNA sequences length to encode weight values. But, if there are edges with their weights $\{1, 2, 100\}$ and the constant factor is 2, then the method is also very inefficient to represent the differences of the weights. Because the lengths of weight sequences are $\{2, 4, 200\}$, one sequence must be very long. For the same reason, it is very inefficient to represent real values. For example, if a graph has the weights of edges such as $\{1.1, 1.2, 10.1\}$, a very long sequence is required to represent 10.1, increasing experiment costs. In addition, the fact that the larger weights are encoded as longer sequences is contrary to the biological fact since the longer the sequence is, the more chances they have to ligate in biomolecules.

To overcome this drawback, we have designed a new encoding scheme. Similar to Narayanan and Zorbalas's second method, we represent edge sequences in two components: link sequences and weight sequences. The difference is that our method uses fixed-length codes. To implement the fixed representation, we represent the weights of edges by varying the amount of A/T pairs and G/C pairs in weight sequences. Generally, the ligation between DNA sequences is influenced by DNA length and the G/C contents [2, 5]. The longer the sequences the more often they get hybridized, thus leading to longer sequences of ligations. Similarly, the more G/C pairs the sequences have, the more probable they get hybridized. The reason is that the hybridizations between the G/C pairs are preferred to those between A/T pairs, since there are 2 hydrogen bonds formed between A and T and 3 hydrogen bonds between G and C, .

The coding scheme is illustrated in Figures 2 and 3. Figure 2(a) shows two vertex sequences, each consisting of 4

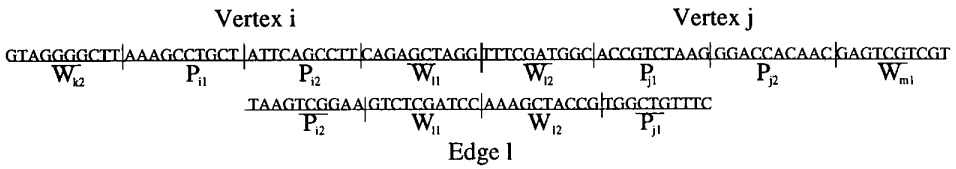

Figure 3: Ligation.

1. Generate the vertex position sequences at random.
2. Generate the edge link sequences controlled by vertex position sequences.
3. Generate the edge weight sequences randomly.
4. Generate the vertex weight sequences according to the edge weight sequences.
5. While (generation $g \leq g_{max}$) do
   (a) Evaluate the fitness of each code.
   (b) Apply genetic operators to produce a new population.
6. Let the best code be the fittest encoding.

Figure 4: Code optimization using a genetic algorithm in Molecular programming.

components: 10-bp weight sequence ($\overline{W_{k2}}$), 10-bp position sequence (P_{i1}), 10-bp position sequence (P_{i2}), and 10-bp weight sequence ($\overline{W_{l1}}$). Figure 2(b) depicts the edge sequence, $V_i \rightarrow V_j$, that also consist of 4 components: 10-bp link sequence ($\overline{P_{i2}}$), 10-bp weight sequence (W_{l1}), 10-bp weight sequence (W_{l2}), and 10-bp link sequence ($\overline{P_{j1}}$). The polarity of edge codes is the opposite (3' - 5') to vertex codes. The position sequences represent a specific vertex; the weight sequence denotes a weight value in an edge. In particular, the position sequence (P_i, P_j) in the vertex represents a specific vertex; the position sequences ($\overline{P_{i2}}$, $\overline{P_{j1}}$) in the edge connects the two vertices.

The weight sequences describe the proportion of edge weights, so our coding scheme can express the real value weights. The way to describe the edge weights is explained in the fitness function.

The weight sequence ($\overline{W_{k2}}$, $\overline{W_{l1}}$) in the vertex makes the weight sequence in the edge work. Only if edge sequences have the weight sequences, the weight sequences are useless because the weight sequences does not have effect on the ligation. Figure 3 illustrate the process of ligation. As explained in figure, the weight sequences are used to ligate.

3 Molecular Programming for Solving TSP

Molecular programming for solving TSPs consists of two parts: a genetic algorithm for optimization of DNA codes and a molecular algorithm for simulating the DNA computing process.

3.1 Genetic Code Optimization

The algorithm is summarized in Figure 4. We illustrate the algorithm using the codes in Figure 2 as an example. In Step 1, the vertex position sequences, P_{i1} and P_{i2} for all i, are randomly generated.

In Steps 2 and 3, the whole edge sequences are designed. For the case of $V_i \rightarrow V_j$, after designing link sequences $\overline{P}_{i2}$, $\overline{P}_{j1}$ and the weight sequence $W_l = (W_{l1}, W_{l2})$ and then they are combined. In Step 4, the vertex weight sequence is designed in a similar way; vertex weight sequences are generated by the edge weight sequences, and then are combined position sequences and weight sequences. In Step 5, the amount of G/C contents in edge sequences is optimized by a genetic algorithm. This is done so that the edges with smaller weights have more G/C contents and thus have higher probability of being contained in the final solution.

Fitness function is defined to promote the paths formed with lower costs (path lengths) so that the minimum cost path could be found. A simple method to do is to count the number of hydrogen bonds in the sequence. For example, the weight value of 20 can be encoded with 20 hydrogen bonds. However this scheme cannot encode real values. To represent real values, we consider the relative number of hydrogen bonds for the weight sequence over the entire sequence. More formally, let N_{e_i} be a function[1] of the number of hydrogen bonds in edge i, S_h the total number of hydrogen bonds in all edges, W_{e_i} the weight of edge i, and S_w the summation of weight values. Then the fitness function is defined as

[1]This function returns the inverse of the number of hydrogen bonds, for example, it returns 2 for G and C, and 3 for A and T.

1. *Encoding*: determine the code sequence using Genetic Code Optimization.
2. While (cycle $c \leq c_{max}$) do
   (a) *Synthesis*: Produce candidate solutions by molecular operators.
   (b) *Separation*: Filter out infeasible solutions by laboratory steps.
3. Keep only those paths that begin with V_{in} (starting vertex) and end with V_{in}.
4. If the graph has n vertices, then keep only those paths that enter exactly $n+1$ vertices.
5. Keep only those paths that enter all of the vertices of the graph at least once.
6. Select the path that contains the largest amount of G/C pairs.

Figure 5: The molecular algorithm for solving TSP in Molecular programming.

$$F_i = \begin{cases} \left| \frac{N_{e_i}}{S_h} - \frac{W_{e_i}}{S_w} \right| & \text{if } \left| \frac{N_{e_i}}{S_h} - \frac{W_{e_i}}{S_w} \right| \geq \theta, \\ 0 & \text{otherwise,} \end{cases} \quad (2)$$

where the threshold value θ is determined by experiments.

The fitness function also takes into account the possibility of reaction errors in the lab steps. The possible errors are checked by simulating the chemical reactions involved with bio-lab steps. The following three illegalities are checked:

1. Ligation of vertex sequences with unacceptable edge sequences,
2. Ligation of vertex sequences that should not be ligated,
3. Unligated vertex sequences that should be ligated.

If an illegal ligation is found, the fitness value will be increased. Therefore, the lower fitness value is better one.

3.2 Molecular Algorithms

The molecular algorithms adopted in this paper the same as the iterative version of molecular programming described in [12]. The iterative molecular algorithm (IMA) iteratively evolves fitter sequences rather than simply filtering out infeasible solutions. This procedure is summarized in Figure 5.

Step 1 is the code optimization process, as described in the previous section. A genetic algorithm is used to assign the best DNA sequences for vertices and edges in the graph. The G/C pairs in the value sequences as well as in the link sequences are adapted to design as reliable codes as possible.

In Step 2, DNA sequences are ligated at random to generate all the possible paths. The edges are selected according to the amount of G/C pair contents and then random paths are created. Because the Iterative Molecular Algorithm (IMA)is used, the paths that have no possibility of being the solution are deleted. The IMA evolves the DNA molecules through synthesis and separation steps. This has an effect of filtering out the wrong solutions in advance and the simulation is accelerated. The deletion criterion is as follows:

1. The path started from V_{in} and ended at V_{in}, but the path length is not $n+1$.
2. The length of the path is longer than $n+1$.

In Step 3, since the starting vertex and the finishing vertex are the same as in the TSP, the paths that only began from V_{in} and ended at V_{in} are selected. Then, the paths that visited exactly $n+1$ vertices are selected. This is the constraint to be satisfied by the definition of TSP: As the graph has n vertices and the finishing vertex is the starting one, the length of path is $n+1$. Finally, the path that has the most G/C contents is chosen as a solution, which represents the minimum cost path.

Since the TSP is similar to the HPP, some steps of the molecular algorithm is the with the algorithms solving HPP [1, 12]. Note, however, that we have additional steps 1 and 2 for code optimization and fitting of good codes and step 6 necessary for TSP using our encoding schemes.

4 Simulation Results

Simulations have been performed on the graph depicted in Figure 1. The parameter values used for the simulations are given in Table 1. Genetic algorithm parameters for code optimization are listed in Table 2. The elitist selection was used. A rather higher mutation rate was applied to give more pressure of variation in search for good DNA codes.

Tables 3 and 4 respectively show the vertex sequences and edge sequences found by the code optimization. Notation $1 \rightarrow 0 \rightarrow 2$ in row 2 of Table 3 means that vertex 0 has in-edge from vertex 1 ($1 \rightarrow 0$) and out-edge to vertex 2 ($0 \rightarrow 2$). Note that the edge sequences with high weights, such as $0 \rightarrow 2$, $0 \rightarrow 3$, and $2 \rightarrow 3$ have more A/T pairs than G/C pairs. Similarly, the edge sequences with low weights, such as $1 \rightarrow 2$ have more G/C pairs than A/T pairs. The link sequences shown in Table 4 contain almost the same number of A/T pairs and G/C pairs. Thus, the weight sequences control the edge weights, and the link sequences have little effect on representing weight values.

Figure 6 depicts the trend of fitness change in code optimization. A fast decrease in fitness is observed until 20 generations, after which the decrease is very slow. This indicates that G/C contents are controlled relatively easily to a certain level, but the zero-error sequence for a specific threshold seems very difficult to find.

Table 1: **Molecular algorithm parameters for molecular7 programming of TSP.**

Parameters	Values
Task	TSP
Pool size	1,000,000
Reaction time	50,000,000
Max cycle	10
Code length for a link sequence	20
Code length for a weight sequence	20
Hybridization error rate	0.003
Ligation error rate	0.003

Table 2: **Genetic algorithm parameters for optimization of DNA codes for molecular programming of TSPs .**

Parameter	Value
Population size	500
Max generation	500
Crossover rate	0.5
Mutation rate	0.4
Ligation error rate	0.001
Threshold	0.001
Selection method	Roulette Wheel

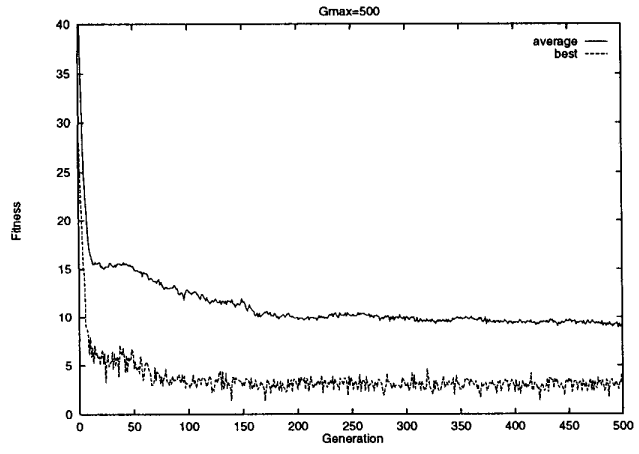

Figure 6: Generation vs. fitness values for code optimization.

The solution for this particular graph is $(V_1 - E_0) \rightarrow (V_2 - E_4) \rightarrow (V_3 - E_8) \rightarrow (V_4 - E_9) \rightarrow (V_1)$. If we represent the sequence in "vertex sequence - weight sequence $\rightarrow$ vertex sequence" form, the solution sequence is

$$
\begin{aligned}
&(CTGATAGTAGCAATTTTTTT \\
&-CTCCGGCCCCCGGGCCGCTG) \rightarrow \\
&(GGAATGACTATGAGGCGAGC \\
&-CCGGCGCGGGCGCCGCCCCG) \rightarrow \\
&(TGCTGATATAAATAGAGAGG \\
&-ACAATATTTTATATATATAT) \rightarrow \\
&(GGACTAATAGAGTTACTCGT \\
&-TAATTTTTTTTAATTTTTAA) \rightarrow \\
&(CTGATAGTAGCAATTTTTTT)
\end{aligned}
\tag{3}
$$

Table 5 shows the weight values in terms of the number of hydrogen bonds in the DNA sequence for the solution. It can be observed that the edges with large weights have a small number of hydrogen bonds, and the edges with small weights have a large number of hydrogen bonds. In addition, the difference of the weights is reflected in the number of hydrogen bonds. This verifies the effectiveness of the proposed encoding scheme.

For comparison, we also simulated Narayanan and Zorbalas' second algorithm with the constant factor $k = 2$. We used the same parameter values as in Table 1 and also used the code optimization method for both algorithms. While we have found 4 solutions out of ten runs, the method of Narayanan and Zorbalas's found no solution for all runs. Their approach ligates the DNA molecules randomly, but our approach takes advantage of the ligation preferences. Therefore, our approach can find the solution more reliably. It should, however, be mentioned that our simulation on NACST used a limited pool size and a limited reaction time. When these parameter values are unlimited as like in actual bio-experiments, both encoding methods will find solutions (if any).

5 Conclusion

We presented a weight encoding scheme for molecular programming and demonstrated its performance on traveling salesman problems. In this encoding, the relative values of G/C contents against A/T contents are taken into account to represent real-valued weights of the edges in the graph. Since G/C pairs have 3 hydrogen bonds and A/T pairs 2, we can control the probability of ligations by balancing them. We have shown that the method is effective for reliable DNA computing applied to TSP problems. The method can easily be modified to use in other graph problems in which edges are associated with real-valued costs. The work in progress is to scale the encoding scheme to large-size problems.

Table 3: **Vertex sequences: Weight and position sequences are shown separately.**

Index	from → V → to	Weight	Position	Weight
1	1 → 0 → 1	GGAGGGGCAA	CTGATAGTAGCAATTTTTTT	GAGGCCGGGG
2	1 → 0 → 2	GGAGGGGCAA	CTGATAGTAGCAATTTTTTT	CAACCCGTTA
3	1 → 0 → 3	GGAGGGGCAA	CTGATAGTAGCAATTTTTTT	AGTATTTAAT
4	2 → 0 → 1	TATATAAATT	CTGATAGTAGCAATTTTTTT	GAGGCCGGGG
5	2 → 0 → 2	TATATAAATT	CTGATAGTAGCAATTTTTTT	CAACCCGTTA
6	2 → 0 → 3	TATATAAATT	CTGATAGTAGCAATTTTTTT	AGTATTTAAT
7	3 → 0 → 1	ATTAAAAATT	CTGATAGTAGCAATTTTTTT	GAGGCCGGGG
8	3 → 0 → 2	ATTAAAAATT	CTGATAGTAGCAATTTTTTT	CAACCCGTTA
9	3 → 0 → 3	ATTAAAAATT	CTGATAGTAGCAATTTTTTT	AGTATTTAAT
10	0 → 1 → 0	GCCCGGCGAC	GGAATGACTATGAGGCGAGC	TATTTAGCGG
11	0 → 1 → 2	GCCCGGCGAC	GGAATGACTATGAGGCGAGC	GGCCGCGCCC
12	0 → 1 → 3	GCCCGGCGAC	GGAATGACTATGAGGCGAGC	AACTAATTAA
13	2 → 1 → 0	GGGGCCGCCC	GGAATGACTATGAGGCGAGC	TATTTAGCGG
14	2 → 1 → 2	GGGGCCGCCC	GGAATGACTATGAGGCGAGC	GGCCGCGCCC
15	2 → 1 → 3	GGGGCCGCCC	GGAATGACTATGAGGCGAGC	AACTAATTAA
16	3 → 1 → 0	CGGGGCACGC	GGAATGACTATGAGGCGAGC	TATTTAGCGG
17	3 → 1 → 2	CGGGGCACGC	GGAATGACTATGAGGCGAGC	GGCCGCGCCC
18	3 → 1 → 3	CGGGGCACGC	GGAATGACTATGAGGCGAGC	AACTAATTAA
19	0 → 2 → 0	TAATAAATAT	TGCTGATATAAATAGAGAGG	CTATTTAATA
20	0 → 2 → 1	TAATAAATAT	TGCTGATATAAATAGAGAGG	GGGGGGCCGG
21	0 → 2 → 3	TAATAAATAT	TGCTGATATAAATAGAGAGG	TGTTATAAAA
22	1 → 2 → 0	GCGGCGGGGC	TGCTGATATAAATAGAGAGG	CTATTTAATA
23	1 → 2 → 1	GCGGCGGGGC	TGCTGATATAAATAGAGAGG	GGGGGGCCGG
24	1 → 2 → 3	GCGGCGGGGC	TGCTGATATAAATAGAGAGG	TGTTATAAAA
25	3 → 2 → 0	AAAAAATGTA	TGCTGATATAAATAGAGAGG	CTATTTAATA
26	3 → 2 → 1	AAAAAATGTA	TGCTGATATAAATAGAGAGG	GGGGGGCCGG
27	3 → 2 → 3	AAAAAATGTA	TGCTGATATAAATAGAGAGG	TGTTATAAAA
28	0 → 3 → 1	ATTTTTTATT	GGACTAATAGAGTTACTCGT	ATTAAAAAAA
29	0 → 3 → 2	ATTTTTTATT	GGACTAATAGAGTTACTCGT	TGGTTAATTG
30	0 → 3 → 3	ATTTTTTATT	GGACTAATAGAGTTACTCGT	ACATATATAT
31	1 → 3 → 1	TGGGCCCGAA	GGACTAATAGAGTTACTCGT	ATTAAAAAAA
32	1 → 3 → 2	TGGGCCCGAA	GGACTAATAGAGTTACTCGT	TGGTTAATTG
33	1 → 3 → 3	TGGGCCCGAA	GGACTAATAGAGTTACTCGT	ACATATATAT
34	2 → 3 → 1	TATATATATA	GGACTAATAGAGTTACTCGT	ATTAAAAAAA
35	2 → 3 → 2	TATATATATA	GGACTAATAGAGTTACTCGT	TGGTTAATTG
36	2 → 3 → 3	TATATATATA	GGACTAATAGAGTTACTCGT	ACATATATAT

Table 4: **Edge sequences: Link and weight sequences are shown separately.**

Index	Link	Weight	Link
0	GTTAAAAAAA	CTCCGGCCCCCGGGCCGCTG	CCTTACTGAT
1	GTTAAAAAAA	GTTGGGCAATATTATTTATA	ACGACTATAT
2	GTTAAAAAAA	TCATAAATTATAAAAAATAA	CCTGATTATC
3	ACTCCGCTCG	ATAAATCGCCCCTCCCCGTT	GACTATCATC
4	ACTCCGCTCG	CCGGCGCGGGCGCCGCCCCG	ACGACTATAT
5	ACTCCGCTCG	TTGATTAATTACCCGGGCTT	CCTGATTATC
6	TTATCTCTCC	GATAAATTATATATATTTAA	GACTATCATC
7	TTATCTCTCC	CCCCCCGGCCCCCCGGCGGG	CCTTACTGAT
8	TTATCTCTCC	ACAATATTTTATATATATAT	CCTGATTATC
9	TCAATGAGCA	TAATTTTTTTTAATTTTTAA	GACTATCATC
10	TCAATGAGCA	ACCAATTAACGCCCCGTGCG	CCTTACTGAT
11	TCAATGAGCA	TGTATATATATTTTTTACAT	ACGACTATAT

Table 5: **Weight values compared to the number of hydrogen bonds in the solution sequence.**

Index	from → to	weight values	hydrogen bond counts
0	$V_0 \rightarrow V_1$	10	103
1	$V_0 \rightarrow V_2$	18	89
2	$V_0 \rightarrow V_3$	20	86
3	$V_1 \rightarrow V_0$	10	102
4	$V_1 \rightarrow V_2$	6	110
5	$V_1 \rightarrow V_3$	12	99
6	$V_2 \rightarrow V_0$	18	89
7	$V_2 \rightarrow V_1$	6	108
8	$V_2 \rightarrow V_3$	18	89
9	$V_3 \rightarrow V_0$	20	88
10	$V_3 \rightarrow V_1$	12	100
11	$V_3 \rightarrow V_2$	18	89

Acknowledgments

Thanks to Sahng-Yun Hahn of Artificial Intelligence Lab (SCAI) and Kwan-Ho Chung of The Institute for Molecular Biology and Genetics, both at Seoul National University, for discussions and comments on the draft versions of this paper. This research was supported by the Seoul National University Research Foundation.

Bibliography

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

[2] E. T. Bolton and B. J. McCarthy, A general method for the isolation of RNA complemetary to DNA, *Proc. Natl. Acad. Sci. USA* **48**: 1390, 1962.

[3] T. Bui and B. Moon, A New Genetic Approach for the Travelling Salesman Problem, *Proc. 1994 IEEE Int. Conf. on Evolutionary Computation*, pp. 7-12, 1994.

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-completeness*, W. H. Freeman and company, 1979.

[5] M. A. Innis, D. H. Gelfand, J. J. Sninsky, and T. J. White, *PCR protocols - A Guide to Methods and Applications*, Academic Press, 1989.

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

[7] R. Lipton and E. Baum, Eds. *DNA Based Computers.* DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. **27**, American Mathematical Sociery, 1996.

[8] C.C. Maley, DNA Computation: Theory, Practice, and Prospects, *Evolutionary Computation*, **6(3)**:201-229, 1998.

[9] A. Narayanan and S. Zorbalas, DNA algorithms for computing shortest paths, *Genetic Programming 1998*, Koza, J. R. *et al.* (eds.), Morgan Kaufmann, pp. 718-723, 1998.

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

[11] C. L. Valenzuela and L. P. Williams, Improving Simple Heuristic Algorithms for the Travelling Salesman Problem using a Genetic Algoritm, *Proc. of the Seventh Int.Conf. on Genetic Algortihms*, pp. 458-464, 1997.

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

[13] B.T. Zhang and S.Y. Shin, Code Optimization for DNA Computing of Maximal Cliques, *Advances in Soft Computing - Engineering Design and Manufacturing*, Sringer-Verlag, 1998.