Minimum Finding with DNA Computing *

Chie-Yao Hsu, Chang-Biau Yang† and Kuo-Si Huang
Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan
†cbyang@cse.nsysu.edu.tw

Chia-Ning Yang
Department of Medical Radiation Technology
I-Shou University, Kaohsiung, Taiwan

Abstract

Recently, DNA computing is one of powerful tools that can be designed for solving NP-complete problems. The powers of DNA computing are that it has great ability of massive data storage and it can process those data in parallel. Some of hard problems, such as the traveling salesperson problem and the Hamiltonian cycle problem, have been solved with the brute force method in DNA computing. After DNA computing is performed, all feasible solutions for the problem are stored implicitly in the tubes. However, the correct answer may not be extracted or reported easily, because that the concentration of the correct solutions might be lower.

In this paper, we introduce some well-known experiment methods in DNA computing and solve the minimum finding problem with DNA computing. Our idea is based on the broadcasting scheme in a parallel system, which is a randomized scheme. In our method, some pilot numbers are selected randomly. Then, the numbers greater than one of pilot numbers may be deleted, and the numbers less than one of the pilot numbers may be amplified. This procedure is repeated until some predefined iterations are reached. Our method allows the chemical reaction error. Thus, our method is fault tolerant. Our idea can also be applied to many other DNA problem solvers.

Key words: molecular computing, DNA computing, minimum finding

*This research work was partially supported by the National Science Council of the Republic of China under contract NSC-91-2213-E-110-022.

1 Introduction

Briefly, DNA computing is that we encode the input and output data as the deoxyribonucleic acid strands (DNA strands) and handle the DNA strands with bio-chemical operations for solving hard problems in the domain of algorithms [4, 8, 11]. These bio-chemical operations are almost accomplished with tubes or DNA chips as the tool of the chemical reactions. By following these operations, we can get the logical or arithmetic solutions with DNA computing. By using the DNA strands, DNA chips or similar tools for solving the computational problems, we usually term these methods as *DNA computing*, *molecular computing* or *bio-computing* [7, 9].

In 1994, Adleman solved the Hamiltonian path problem (HPP) with DNA strands successfully in a laboratory [1]. Since then, many scholars made a great effort for DNA computing. The gate for solving the computational problems with DNA computing has been opened. There are two main differences between DNA computers and traditional computers. The first important difference is that it only takes few amount of DNA strands to store the data that we have encoded. For instance, there are almost 10^{20} DNA molecules in 1μmol. If we use one nucleotide to represent one bit, 1μmol DNA molecules can represent 10^{20} bits. The second difference is that bio-chemical reactions take place simultaneously in one tube. So, we can deal with the data and operations in parallel. These two differences are also the advantages of DNA computing compared to a traditional computer system.

In DNA computing, four characters `A`, `G`, `C`, and `T` denote four different nucleotides adenine, guanine, cytosine, and thymine, respectively. Some-

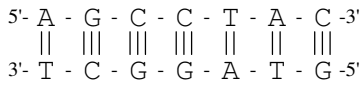

Figure 1: Hybridization: the connection of A and T with two hydrogen bonds, and the connection of G and C with three hydrogen bonds.

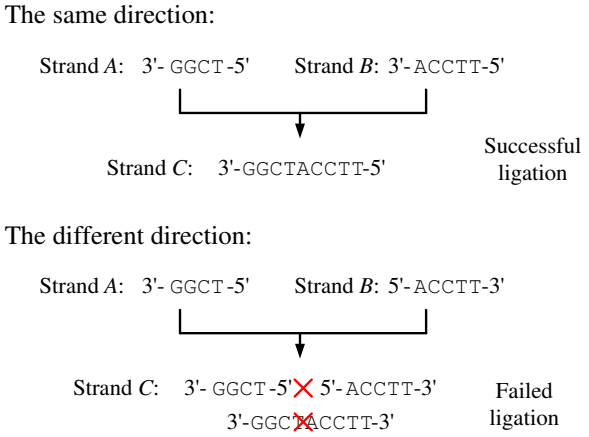

Figure 2: Ligation for strands A and B with different directions.

times we may need some artificial nucleotides for implementing DNA computation. We call nucleotides A, G, C, and T and artificial nucleotides as *bases*. Among these bases, A uses two hydrogen bonds to connect to its complement T, and G uses three hydrogen bonds to connect to its complement C naturally when they are *anti-parallel*, as shown in Figure 1. *Anti-parallel* means that the direction of one DNA strand is 5' $\rightarrow$ 3', and the direction of the other one is 3' $\rightarrow$ 5'. Figure 1 gives an illustration that these two anti-parallel nucleotides will anneal to each other. One can generate a strand with these bases which are in the same direction (5' $\rightarrow$ 3'), so that these bases can ligate to each other without nicks. If we want to ligate several DNA strands, they must be in the same direction, as shown in Figure 2.

DNA computing possesses the abilities of parallel computing. Different data can be encoded into DNA strands. Every DNA strand can be processed with the bio-chemical operations that we designed. We can take those bio-chemical operations as computational processes of a traditional computer. For each DNA strand, it can be complemented to its complementary DNA strand. Then the two single stranded DNA strands will hybridize together, and we can get one double-stranded DNA sequence. If the bio-chemical operations in the tube can be considered as logical and arithmetic processes of the traditional computer, the way which all DNA strands complement with their complementary DNA strands in the tube is similar to the parallel logical or arithmetic processing in the traditional computer. We can take it as the multiple processors to execute the same instructions in the parallel computer. In the classes of the computational characteristic, we can classify the DNA computing into SIMD (Single Instruction Multiple Data) [2, 5].

In this paper, we develop a novel method to solve the minimum finding problem with DNA computing. We imitate the broadcasting way in a parallel system and the thought of concentration to solve the minimum finding problem. Our algorithm uses these ideas to achieve the ability of the fault tolerance in DNA computing.

The rest of this paper is as follows. We introduce some useful bio-chemical experiment methods in DNA computing and the minimum finding problem in Section 2. Section 3 gives our main idea and the method for solving the minimum finding problem with DNA computing. Section 4 describes the details of DNA encoding forms and the generation of feedbacks. The simulation and the analysis of our algorithm on computer are shown in Section 5. Finally, we give some conclusions in Section 6.

2 Prelimilary

We need a lot of bio-experiment methods and related knowledge for solving hard problems in the traditional computer [3, 12, 14]. We shall describe some well-known bio-experiment methods [15] as follows.

1. **Hybridization and annealing** Watson-crick complementarity describes the rule of hybridization clearly. In the converse direction, nucleotides will connect to their complementary nucleotides with hydrogen bonds. In DNA computing, $c(S)$ or $\overline{S}$ means the reversely complementary single DNA strand S. If two DNA strands are complementary single-stranded, the two single strands of DNA will bond to form a double strand of DNA. This is the so-called *hybridization* or *annealing* as shown in Figure 3.

2. **Synthesis and PCR** *Synthesis* is to manufacture the DNA strands that we desire. One

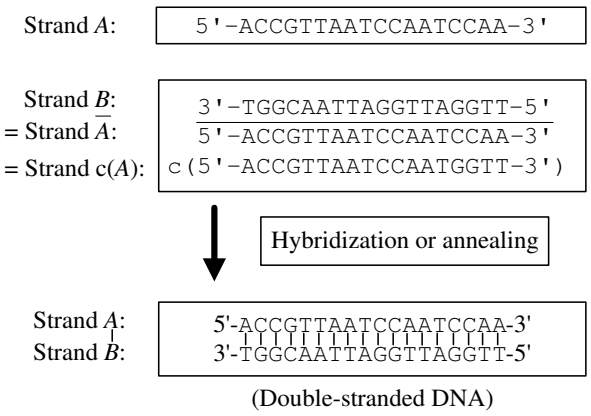

Figure 3: An example of single strand of DNA, its complementary strand, and their double strand of DNA. The complementary single-stranded DNA strands A and B in the anti-parallel direction can anneal together to form a double-stranded DNA strand.

can synthesize any kind of DNA strands in a laboratory, and we can amplify the DNA strands by *PCR* (Polymerase Chain Reaction). The principle of PCR is to take a single strand as a template. We can produce another identical DNA strand with the Watson-Crick complementarity. PCR needs lots of primers and nucleotides to start the process. When primers anneal to the template, the polymerase can start to duplicate the DNA strands. Then, the single strands will anneal the nucleotides to form the double-stranded one. Then separate the double-stranded DNA strands to two single-stranded DNA strands, Repeatedly applying the PCR procedure, we can obtain a lot of copies of the DNA strand that we desire to amplify. Briefly, PCR is a kind of great technologies to duplicate DNA strands.

3. **Primer extension** *Primer extension* is the first step in PCR. Before duplicating a DNA strand, we must pour a small DNA segment which is the complement to one part of the DNA strand which we want to duplicate as the complementary. And it will cause a series of reactions that the other DNA segments will bond the other part of the single strands. Just like the train, the railway carriages move by following the locomotive. In PCR, primers play the role of the locomotives.

4. **Restriction enzyme digestion** There are some specified ordered sequences in a double-stranded DNA strand, restriction enzymes can be applied to break the DNA strands into segments by cutting these specified ordered sequences. The specified ordered sequences are called *restriction sites*. In other words, restriction enzymes can recognize and attack the corresponding restriction sites. But each kind of restriction enzyme can attack only the unique restriction site. The DNA strands of restriction sites must be double-stranded. The restriction enzymes only work when there are no nicks in the restriction sites.

5. **Ligation** *Ligation* means linking two DNA strands with nicks, and making them into a complete DNA strand. Without ligation, we cannot use restriction enzymes to cut restriction sites with nicks in the DNA strand because there may exist nicks.

6. **Gel electrophoresis** *Gel electrophoresis* is an indispensably important method of bio-experiments. We can classify lots of DNA strands easily by simply using the gel electrophoresis. We first put the DNA strands into the gel, and place the gel in the two extremities of the electric field. There are lots of negative electrons in the DNA strands, so the DNA strands will move to the positive side of the electric field.

7. **Methyled DNA** The DNA strands with *methyl-group* will make the original restriction site lose its efficacy. It means that the restriction sites of the methyl-group DNA strands can't be attacked by the restriction enzymes [6]. However, the methyl-group DNA strands can still have the abilities of hybridization and annealing.

3 Minimum Finding with DNA Computing

We can encode the input data and represent them as DNA sequences. Each sequence represents a numerical value. We want to find the smallest value among a set of values represented by DNA strands. There are some limits here. First, we represent the numerical data as binary data. Second, the encoding form of each bit in DNA strands is unique. For instance, the encoding form of the first bit 1 in the DNA strand (1010) is different from bit 3 with value 1. Note that we count

the right most position as bit 0, whose left is bit 1, and so on. We want to find out the solution which the value is minimum among these encoded DNA strands. Our algorithm is based on the broadcasting way to find out the minimum value in parallel computers [10, 13]. The main idea is that each node of the parallel system stores a value, and we want to find the minimum among those nodes. If there are n nodes, we first choose a node x randomly. Node x will broadcast its value to other nodes in the parallel system. After broadcasting, all other nodes will receive the value of x. The node whose value is greater than x would lose the chance to broadcast its value in next round. Only the nodes whose values are less than x can broadcast their values. In the next round, we randomly choose another node to broadcast its value. This procedure is executed repeatedly. Finally, we can obtain the minimum if no node can broadcast its value furthermore. If the number of nodes in the parallel system is n, it takes at most n rounds to find out the minimum. In average, $O(\log n)$ rounds are required [10, 13].

Usually, the ways of solving problems with DNA computing use the brute force method. Though all feasible solutions are in the tubes, the correct answers still might not be found out absolutely. The reason is that the concentration of the correct solutions might be lower. So, if the concentration of the correct answers can be increased, they can be found more easily. Our main idea is trying to increase the concentration of the correct solutions and decrease the concentration of other solutions.

If we implement above broadcasting algorithm with DNA computing and blend with the idea of the concentration, we believe it can improve the fault tolerance of DNA computing and the efficacy of finding out the correct solutions will be increased. The fault tolerance of DNA computing is to increase the number of correct solutions such that they are more than other solutions in tubes. In DNA computing, we encode our data with DNA strands. Each strand stands for a value. Each value is represented with a DNA strand in binary form. The encoding form of each bit is different to each other. So there would be no interaction in different bits. And different bit positions are connected together with the restriction sites. Every strand will produce a feedback based on its value as shown in Figure 4. The feedbacks will anneal to the strands whose holding values are greater than that value represented by the original strand. The algorithm is as follows.

Algorithm: Minimum Finding

Input: A set A of binary numbers encoded by DNA strands.

Output: The minimum number in set A.

Step 1: Randomly choose p percentage of DNA strands as X from solution pool A.

Step 2: Produce feedbacks according to X.

Step 3: Pour feedbacks into A.

Step 4: Resupply A with the DNA strands of the original concentration.

Step 5: Purify pool A to get a new pool A'. Let A=A', where A' denotes those strands whose values are less than X.

Step 6: Repeat Step 1 to Step 5 until the concentration of set A is less than the predefined threshold.

Suppose each number has n binary bits. Let $B = b_{n-1}b_{n-2}\cdots b_1b_0$ be an input number, where each b_i is of value either 0 or 1. The set of feedbacks for B, denoted as $F(B)$, consists of the following elements: $c(b_{n-1}\cdots b_{i+1}1)$ if b_i=0, where $c(b_{n-1}\cdots b_{i+1}1)$ represents the complementary DNA strand of $b_{n-1}\cdots b_{i+1}1$. Here, the bit positions are counted in the way that the rightmost position is 0. For each number, we generate one feedback for each bit with value 0. For example, suppose $B = 0101$ and $D = 1010$. Then $F(B) = \{c(1), c(011)\}$, and $F(D) = \{c(11), c(1011)\}$. As another example, suppose $B = 011001$. Then $F(B) = \{c(1), c(0111), c(01101)\}$. For each DNA strand representing one number B, we randomly produce one feedback in $F(B)$ for it. Since one number B is encoded with many DNA strands with the same form, we can say that we shall produce every feedback in $F(B)$. Figure 5 shows the result after the feedbacks for 0101 are generated.

After the feedbacks annealing to strand A whose value is greater than the original strand, strand A can be attacked by restriction enzymes. So we can separate the strands that are not attacked by enzymes. We can randomly choose the strands that are not attacked by enzymes to generate the feedbacks again and pour the feedbacks back. Again and again, the concentration of the larger value in the original tube will be decreased, as shown in Figure 6. After the final iteration, the DNA strands representing the smaller values will be amplified with PCR. Because the DNA strands

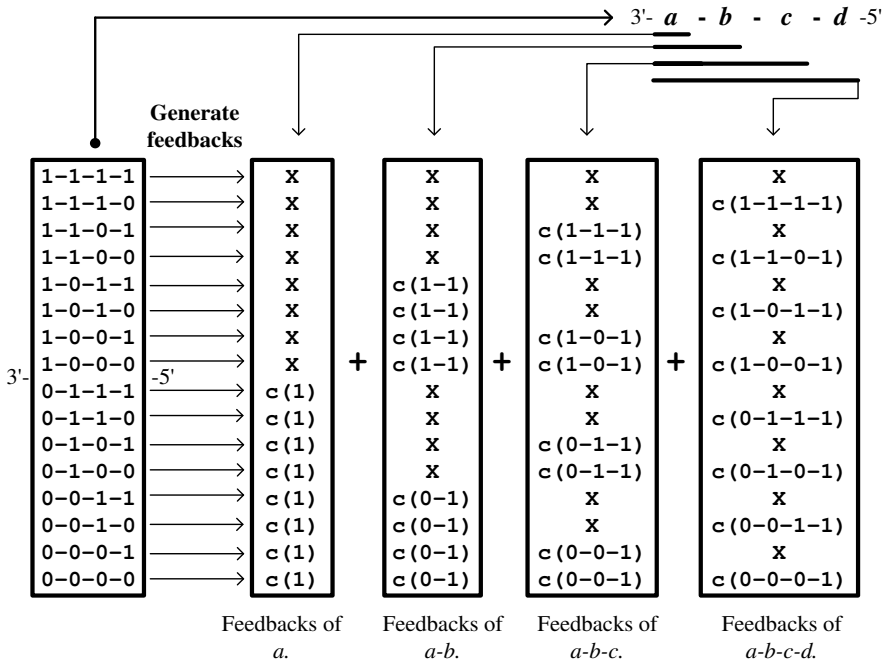

Figure 4: The values of a 4-bit system and their corresponding feedbacks.

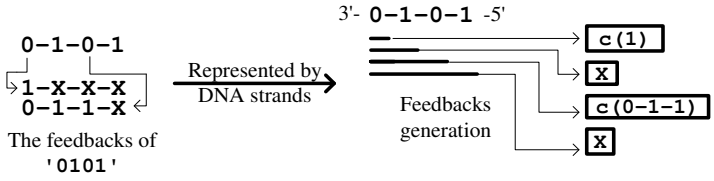

Figure 5: The feedbacks of $'0101'$.

representing the smaller values do not be attacked by enzymes, we can select the smaller values according to its size (n bits). The amount of the DNA strands representing the smaller values will be increased.

In Algorithm Minimum Finding, we select a part of DNA strands and use them for generating feedbacks. After pouring the feedbacks into the original tube, the feedbacks will simultaneously anneal to the DNA strands whose values are greater than the original values. It can be viewed as that the DNA strands can generate feedbacks as the nodes which can transmit their values to other nodes. There is a slight difference between a parallel computing system and a DNA computing system. In a parallel system, a value can be broadcast via a common channel or bus. Thus we need generate only one value for broadcasting. In a DNA system, one value is represented by a DNA strand,

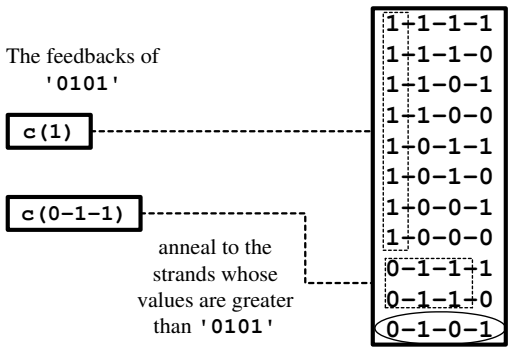

Figure 6: An example of the feedback action. Feedbacks of strand $'0101'$ will anneal to the strands whose values are greater than $'0101'$.

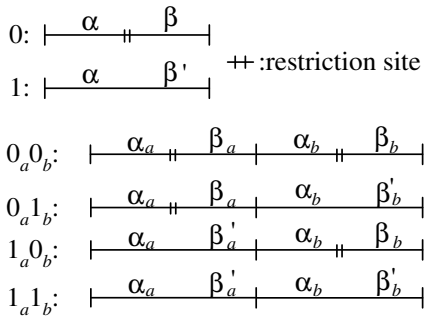

Figure 8: The encoding forms of 1 bit and 2 bits.

which can generate only one feedback. One feedback can anneal to only one DNA strand, which will be attacked (removed) later. That is, a DNA strand can transmit its value to only one DNA strand. In fact, one value is encoded by a lot of DNA strands. Thus, one value can be broadcast by these great amount of encoded DNA strands. Besides, in a parallel system, the way of communication is only one-way, because the node only can broadcast a message or receive a message at the same time. In DNA computing, the way of communication is two-way, each node can receive or broadcast messages at the same time. Figure 7 illustrates an example of our method in global view.

4 The DNA Encoding Forms and Feedback Generation

The encoding form of one binary bit with a DNA segment is divided into parts: α and β (or β'), as shown in Figure 8. α, β and β' consist of different DNA segments and they will not hybridize to each other. Value 0 is composed of α and β. Value 1 is composed of α and β'. Besides, there is one restriction site on value 0, but no restriction site on value 1. So the encoding forms of 0 and 1 are different to each other. Furthermore, the encoding forms of different bit positions with the same value are still different to each other. In the tube, there are a lot of DNA segments that represent the same value. We take some percentage of the DNA sequences for generating feedbacks.

First, if we need n bits to represent one number element, we will generate n kinds of DNA feedback segments. The first kind of feedbacks is generated by the most significant bit. The second kind of feedbacks is generated by the most two significant bits, etc. Suppose B=$b_{n-1}\ b_{n-2}\ \cdots\ b_0$ is an input number element. The ith kind of feedbacks is generated by $b_{n-1}\ b_{n-2}\ \cdots\ b_{n-i}$ if b_{n-i}=0. We use a single DNA strand 0_i, $0 \le i \le n-1$, to denote bit position i with value 0, 1_i to denote bit position i with value 1, and $\overline{0_i}$ to denote the complementary DNA strand of 0_i.

After finishing the generating process for feedbacks, we pour them into the original tube which contains all solutions. Each feedback will hybridize the DNA strand whose value is greater than the feedback. The DNA strands hybridized by feedbacks will become partially double-stranded. Then we use restriction enzymes to digest these double-stranded DNA strands. Thus, they will become shorter (not of full length). By doing so, we call above procedure as one iteration. The algorithm for generating the feedbacks is as follows and the illustrations are shown in Figure 9 and Figure 10.

Algorithm: Generating Feedbacks

Step 1: Divide some percentage of DNA sequences into n tubes (t_{n-1}, t_{n-2}, $\cdots$, t_0). Here, tube t_{n-i} will generate the ith kind of feedback.

Step 2: Pour $\overline{0_i}$, $0 \le i \le n-1$, into tube t_i respectively. Each $\overline{0_i}$ will anneal to 0_i, which is at bit position i of one input number element.

Step 3: In each tube t_i, pour the restriction enzymes to digest the double-stranded DNA segments to cut α and β of bit position i with value 0.

Step 4: In each tube t_i, keep the bit string starting from the most significant bit to bit i of the DNA strand by using primer extension.

Step 5: Pour 1_i, $0 \le i \le n-1$, into tubes t_i respectively. Then the operations, denaturing and annealing are used to reanneal to 1_i for the bit string obtained in Step 4.

Step 6: Using primer extension to make the $\overline{1_i}$ completely.

Note that Step 4 should be omitted when the first kind of feedback is generated in tube t_{n-1} since there is no bit left to bit position $n-1$. The first step in our algorithm is the preprocess of each bit for generating feedbacks. Each number element has n bits, the generation of feedbacks also takes n tubes to implement. It will produce n kinds of feedbacks with different lengths. We can not generate all kinds of feedbacks in the same

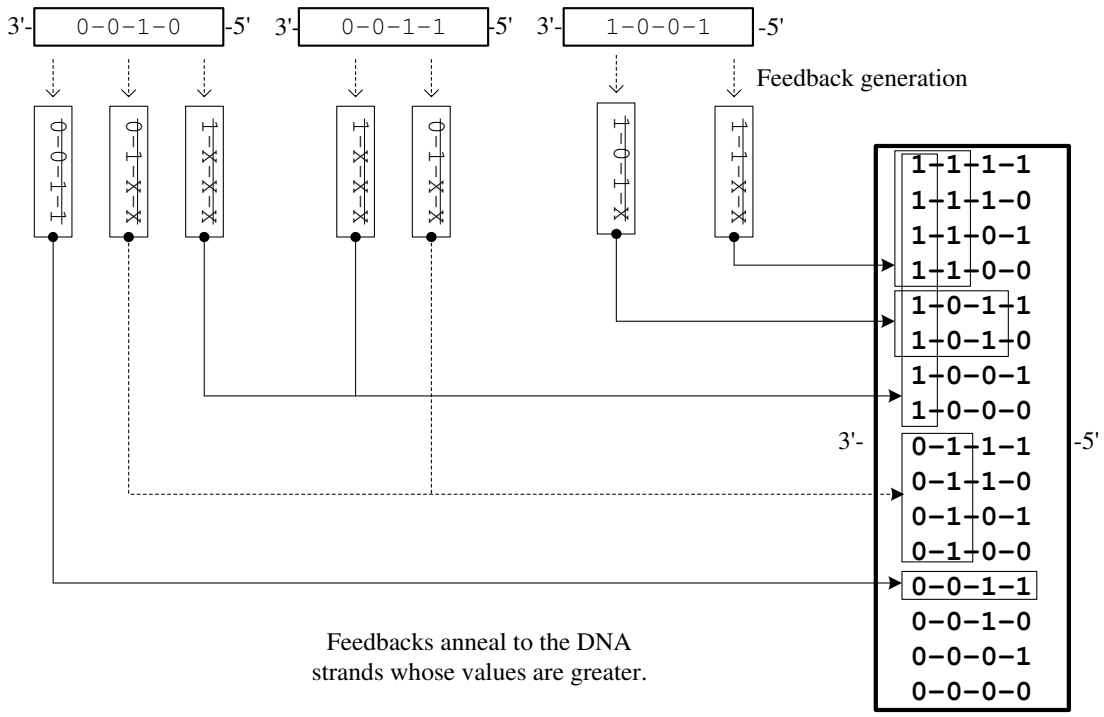

Figure 7: Global view of minimum finding in DNA computing.

tube because there will be more than one digestion point after annealing, and it will obtain a wrong result. The main purpose of the feedback generating algorithm is to transform the DNA strands of value 0 into the DNA strands of value 1 at the same bit position.

Now, we shall give the detailed description for each step. In Step 2, the DNA strands of value $\overline{0}$ hybridize with the corresponding bit positions of DNA strands of value 0. And we can keep the bit string starting from the most significant bit to the corresponding bit position in Step 3 and Step 4. In Step 5, the DNA strands obtained in Steps 3 and 4 will hybridize with the DNA strands of value 1 by denaturing and annealing. After primer extension in Step 6, the transform from value 0 to value 1 is done. The feedback generation is finished. For instance, the DNA strands representing the value '1001' will be transformed into the DNA segment representing the value '101' in tube t_1 and '11' in tube t_2.

5 Simulation and Analysis

We made the simulation in the computer and Table 1 shows the simulation result. There are three data sets, 100, 120 and 140 numbers respectively, in our simulation. Each number is represented by 300 DNA strands repeatedly, the probability for making errors in the feedback generation is assumed to be 1% and 2% respectively. The probability for making errors in the feedback annealing with the original strands is assume to be 1% and 2% respectively. The output data are the concentration of each number. We do the simulation 500 times for each input data set and calculate the average the output data.

The first column of the table represents the iterations, the second column represents the sizes of the sets of the input numbers, and the first row represents the ratio of the concentration of the smallest numbers in that set. We suppose that the percentage of the error situation is small.

For example, the concentration of the smallest number in the set of 100 numbers is very near 100% in the final iteration of the simulation. After each iteration, the percentage of the smallest number is increased. As we can see in the simulation result, our method can really increase the concentration of the better solutions.

According to the minimum finding algorithm with DNA computing, we can analyze it with formula. The formula of the concentration of the

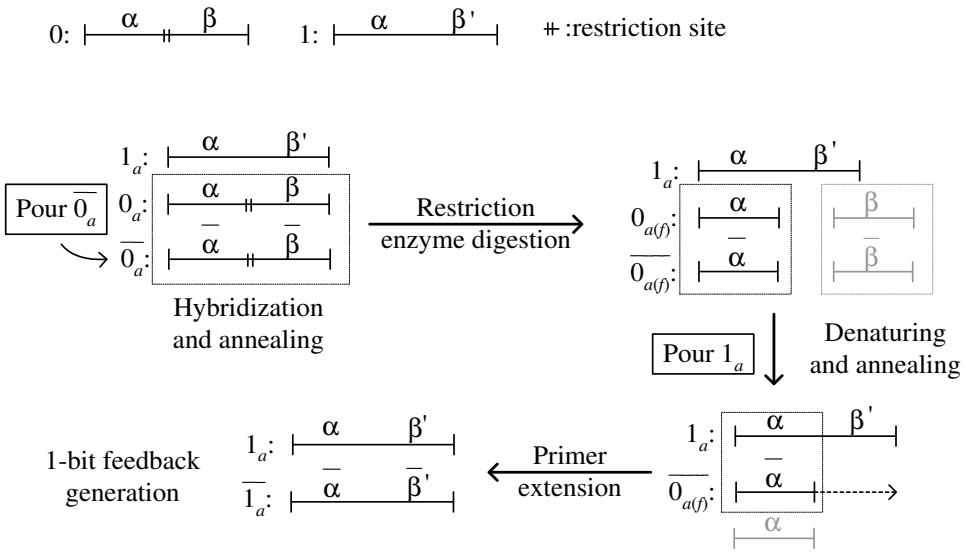

Figure 9: A generation of 1-bit feedback. Here, $0_{a(f)}$ denotes the front subsegment of bit position a with value 0.

Iter.	Input numbers					
	100		120		140	
	Error rates					
	1%	2%	1%	2%	1%	2%
1	1%	1%	0.8%	0.8%	0.7%	0.7%
2	24%	23%	18%	17%	17%	16%
3	42%	41%	33%	32%	32%	31%
4	56%	55%	45%	44%	45%	44%
5	66%	66%	56%	55%	56%	54%
6	76%	75%	64%	63%	63%	62%
7	80%	80%	72%	71%	73%	71%
8	85%	85%	79%	78%	78%	77%
9	92%	90%	82%	82%	81%	80%
10	97%	96%	85%	85%	84%	84%
11	100%	99%	89%	88%	89%	88%
12			92%	92%	93%	92%
13			97%	95%	97%	96%
14			100%	100%	100%	100%

Table 1: The simulation result of our DNA computing algorithm. Each row represents the concentration of the smallest number in the data set. Each column represents the size of the data set.

smallest number is

$$\frac{1}{n(1-P)^i}$$

where n, i, and P denote the size of input data set, the number of iterations, and the percentage of the feedback generating and minus the error rate respectively.

6 Conclusion

In this paper, we propose a randomized minimum finding algorithm with DNA computing. Our goal is to decrease the quantity of bad or wrong solutions in the chemistry tube. We implement the feedback scheme with the help of restriction enzymes. Then, PCR is used to duplicate the solutions with correct size (n bits). So, we can easily divide the solution space into two parts in which one is correct and the other is not correct. Our idea can be used to overcome the chemical reaction errors during the bio-chemical experiments. Besides, it can also be applied in many DNA problem solvers, such as the solution to the traveling salesperson problem or the sum of subset problem. In these problems, the minimum finding is the key issue to find the optimal solution, and the correct solutions are usually not easy to find out. So, if we can increase the quantity of the optimal solutions, it will be more efficient in finding the experiment results. Our algorithm has the ability of fault tolerance, so we can classify our method as a novel strategy for optimization in DNA computing.

In the future work, we may add some strategies, for example, the genetic algorithm and the ant colony system. If we can apply these strategies to DNA computing, the brute force method will not be the main method of DNA computing. Some approximation algorithms will be designed according to these strategies. Therefore, hard problems may be solved more efficiently with DNA computing.

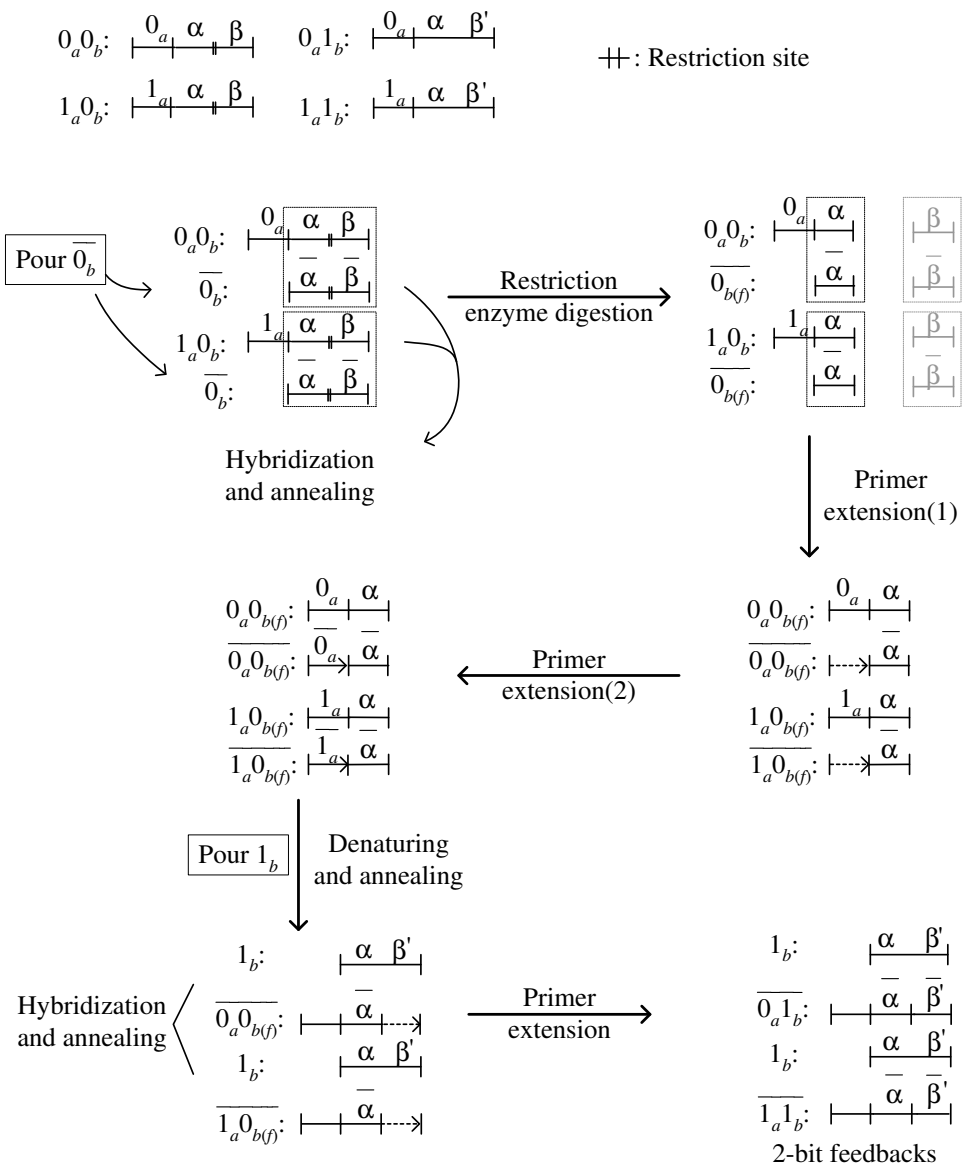

Figure 10: A generation of 2-bit feedback.

References

[1] L. M. Adleman, "Molecular computation of solutions to combinatorial problems," *Science*, Vol. 266, No. 5187, pp. 1021–1024, Nov. 1994.

[2] S. G. Akl, *The Design and Analysis of Parallel Algorithms*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, first ed., 1989.

[3] T. Akutsu, S. Kuhara, O. Maruyama, and S. Miyano, "Identification of genetic networks by strategic gene disruptions and gene overexpressions under a boolean model," *Theoretical Computer Science*, Vol. 298, No. 1, pp. 235–251, 2003.

[4] A. Dove, "From bits to bases: Computing with DNA," *Nature Biotechnology*, No. 9, pp. 830–832, Sep. 1998.

[5] S. P. A. Fodor, "Massively parallel genomics," *Science*, Vol. 277, No. 5324, pp. 393–395, July 1997.

[6] R. H. Garrett and C. M. Grisham, *Biochemistry*. Fort Worth: Saunders College Pub., second ed., 1998.

[7] G. Gloor, L. Kari, M. Gaasenbeek, and S. Yu, "Towards a DNA solution to the shortest common superstring problem," *International Journal on Artificial Intelligence Tools*, Vol. 8, No. 4, pp. 385–400, 1999.

[8] M. Hagiya, "From molecular computing to molecular programming," *DNA Computing, 6th International Workshop on DNA-Based Computers, DNA 2000, Lecture Notes in Computer Science*, Vol. 2054, pp. 89–102, June 2001.

[9] T. Hinze and M. Sturm, "A universal functional approach to DNA computing and its experimental practicability," *In Proceedings of the Sixth DIMACS Workshop on DNA Based Computers, The University of Leiden, Leiden, The Netherlands, The University of Leiden, Leiden, The Netherlands*, pp. 257–266, June 2000.

[10] S. P. Levitan and C. C. Foster, "Finding an extremum in a network," *ISCA '82: Proceedings of the 9th annual symposium on Computer Architecture, Austin, Texas, United States*, pp. 321–325, IEEE Computer Society Press, 1982.

[11] R. J. Lipton, "DNA solution of hard computational problems," *Science*, Vol. 268, pp. 542–545, Apr. 1995.

[12] D. L. Robertson and F. G. Joyce, "Selection in vitro of an RNA enzyme that specifically cleaves single-stranded DNA," *Nature*, Vol. 344, pp. 467–468, Mar. 1990.

[13] S. H. Shiau and C. B. Yang, "A fast maximum finding algorithm on broadcast communication," *Information Processing Letters*, Vol. 60, pp. 81–89, 1996.

[14] W. P. C. Stemmer, "Rapid evolution of a protein by DNA shuffling," *Nature*, Vol. 370, No. 6488, pp. 389–391, Aug. 1994.

[15] H. Y. Wang, C. B. Yang, K. S. Huang, and Y. L. Shiue, "The design of sorters based on DNA for bio-computers," *International Computer Symposium, Workshop on Algorithms and Computational Molecular Biology*, National Dong Hwa University, Hualien, Taiwan, Dec. 2002.