International Journal of Innovative
Computing, Information and Control
Volume **5**, Number **10(A)**, October **2009**

ICIC International ©2009 ISSN 1349-4198
pp. 3045–3057

ERROR-TOLERANT MINIMUM FINDING WITH DNA COMPUTING

Chia-Ning Yang[1], Kuo-Si Huang[2], Chang-Biau Yang[3,*] and Chie-Yao Hsu[3]

[1]Institute of Biotechnology
National University of Kaohsiung
Kaohsiung, 811, Taiwan
cnyang@nuk.edu.tw

[2]Department of Information Management
Shu Zen College of Medicine and Management
Kaohsiung, 821, Taiwan
huangks@szmc.edu.tw

[3]Department of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, 804, Taiwan
*Corresponding author: cbyang@cse.nsysu.edu.tw; hsucy@par.cse.nsysu.edu.tw

Received May 2008; revised October 2008

Abstract. *In the past decade, DNA computing has become one of the powerful approaches to solve a class of intractable computational problems such as Hamiltonian path problem and SAT problem. The power of DNA computing comes from the fact that it has great potential of massive data storage and processing computation over data in parallel. In this paper, an algorithm for solving the minimum finding problem is proposed in theory with a randomized scheme – the broadcasting scheme on the broadcast communication model. Our method is a novel strategy for optimization in DNA computing, since it can tolerate some chemistry reaction and experiment errors. The less errors are, the less the number of required iterations is. A series of computer simulations are also performed and analyzed to demonstrate the feasibility of the algorithm and the designed experiment.*

Keywords: Molecular computing, DNA computing, Minimum finding, Chemistry experiment error

1. **Introduction.** DNA computing uses DNA strands to encode input and output data and handles the strands with biochemical operations for solving hard problems in the domain of algorithms [2, 8, 11, 12, 13, 14, 16, 17, 25]. Compared to the traditional silicon-based computing system, there are two advantages in DNA computers – high capacity of data storage and massive parallelism in computing. For instance, there are about 6×10^{17} DNA molecules in 1 μmol, and if one bit is encoded with one nucleotide, 1 μmol DNA molecules hold 6×10^{17} bits of information, which is much more than the most massive scale 10^{12} bits (Tera bits) of the storage in a computer nowadays. In DNA computing, different data can be encoded into DNA strands, which can be processed with the biomolecular operations such as primer extension, PCR, restriction enzyme digestion, and gel electrophoresis [4, 10, 18, 21, 23]. If these operations in the tube can be regarded as logical and arithmetic processes of the traditional computer system, the interactions of DNA strands in the tube will be similar to the logical and arithmetic processing in a traditional computer. Moreover, because the interactions take place in the tube simultaneously, DNA computing can be regarded as multiple processors to execute the same instructions in a parallel computer. Consequently, instead of treating data one by one,

processing data in such a parallel style certainly accelerates the computing course. In the classes of the computational characteristic, DNA computing can be classified as SIMD (Single Instruction Multiple Data) [3, 9]. In addition to hard problems, topics including information security and data clustering [1, 6, 22] with DNA computing were also of interest lately.

In general, DNA computing often takes the strategy of brute force search in solving problems. In short, a library of all possible answers are considered at very beginning followed by a series of biomolecular laboratory techniques to systematically eliminate the wrong answers and to collect the correct answers. Due to the huge DNA computing power, most researchers focused on solving intractable or NP-hard problems, such as satisfiability, Hamiltonian cycle and traveling salesperson problems. For decision problems, such as satisfiability and Hamiltonian cycle problems with answers either 'YES' or 'NO', the wrong answer elimination scheme can always be applied. However, for the optimization problem, such as the traveling salesperson problem, the extraction of minimum or maximum value should be involved. Thus, the minimum or maximum finding algorithms become essential in the solutions of optimization problems. When applying a new computational model, such as DNA computing, some elementary and yet simple, algorithms like minimum finding, sorting, integer addition, and integer multiplication, should be built in the model to serve as the basis for solving complicated problems. Hereby, we propose an algorithm to find an extreme number among a numerical data pool.

Several novelties are taken into account in this work. First, unlike other DNA computing algorithms directly taking various lengths of DNA-sequences to represent different values, we develop a binary coding scheme on DNA sequence in hope to handle numerical problems with more efficiency, especially for problems covering values in a wide range. Second, we utilize a simple computational model broadcast communication model (BCM) [5, 7, 20, 24] in finding the extrema with DNA-computing, since it has been shown that the minimum (or maximum) finding has a good implementation on BCM [15, 19]. Third, with PCR and portion control techniques our proposed algorithm is designed to tolerate experimental and chemical reaction errors. To demonstrate the error tolerance and the interaction among input numbers in this algorithm, we build a mathematical model on which theoretical analysis is given. Furthermore, we also perform a series of computer simulations of the proposed algorithm to illustrate the feasibility of our method.

2. **Backgrounds.**

2.1. **Minimum finding on the broadcast communication model.**

Our algorithm is based on the broadcasting way to find out the minimum value on the broadcast communication model [15, 19]. The main idea is that each node of BCM stores a value, and our goal is to find the minimum among the given n nodes. At first a node x is chosen indiscriminately and broadcasts its value to other nodes in BCM. After broadcasting, all other nodes will receive the value of x. The nodes with values greater than x will lose the chance to broadcast their values in the following iterations whereas the nodes with values less than x hold the chance to broadcast their values in the latter iterations. In the next iteration, again another node is randomly chosen to broadcast its value. Such a procedure is repeated until the minimum is determined under the circumstance of no node broadcasting its value furthermore. It needs at most n iterations to find out the minimum in the worst case, where n denotes the number of nodes in BCM. In average, $O(\log n)$ iterations are required [15, 19].

2.2. **Minimum finding with DNA computing.**

Taking the advantage of parallelism in DNA computing, we can randomly pick up more than one node from a parallel system

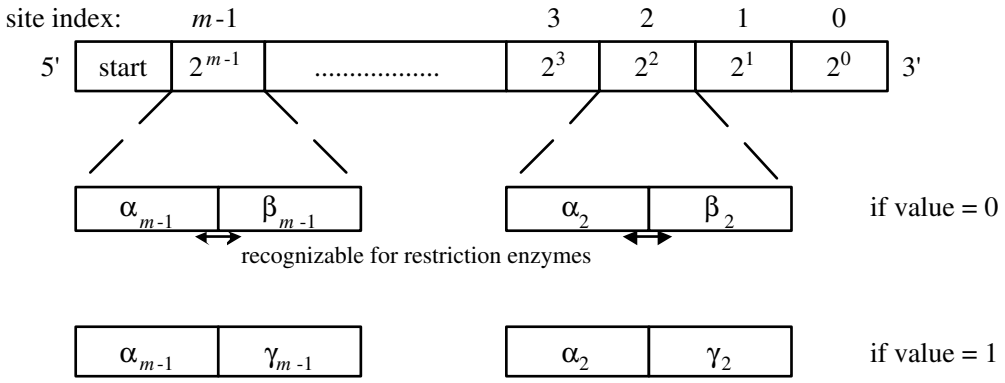

FIGURE 1. The sequence assignment of an m-bit number. Fragment sequences for representing α, β, and γ vary with site i.

of n nodes at a time and broadcast their values simultaneously to exclude many more nodes with greater values. Consequently, this shall greatly speed up the minimum finding process.

To solve a problem with n values, each with m binary bits, n sorts of single-stranded DNA sequences in measurable amount are considered to represent an input data pool. As illustrated in Figure 1, each strand comes with a short segment 'start' at one end and the rest of sequence is divided into m sites where each site can be assigned as value 0 or 1. We count the right most position (the 3' end of DNA sequence) as bit 0, and the left most one (the 5' end of DNA sequence) as bit $m-1$. Sequence assignments for different sites of value 1 or 0 are unique. For example, in the DNA strand for binary number 1010, the encoding form of bit 1 is different from bit 3, although they both are of value 1. Moreover, each sequence assignment for each site of 0 is recognizable by a corresponding restriction enzyme whereas sequence assignments for value 1 are not.

Every strand can produce feedbacks based on its value and these feedbacks function to target other numbers with larger values. Similar to BCM, the whole minimum finding procedure is completed by several iterations and in each iteration a portion of strands are randomly selected to produce feedbacks outside of the data pool. Further, by pouring the feedback to the pool as broadcasting, strands with larger values are discriminated and discarded. With repeated iterations, the pool size shrinks so that the strands with the smallest value gradually stand out. The computing ends when the pool size stays almost the same after a broadcasting iteration.

2.3. **Algorithm: Minimum finding.**

Input:: A set A of n binary numbers, with m bits, encoded by DNA strands.
Output:: The minimum number in set A.
Step 1:: From solution pool A, randomly choose ω portion of DNA strands to form X, $0 < \omega < 1$.
Step 2:: Produce feedbacks according to X's values.
Step 3:: Pour feedbacks into A to eliminate numbers larger than X's.
Step 4:: Purify pool A to get a new pool A'. Let A=A', where A' denotes those strands with values less than or equal to X's.
Step 5:: Repeat Steps 1 to 4 until the quantity of A remains nearly unchanged.

Suppose each number has m binary bits and the bit positions are indexed as mentioned earlier and therefore each input number can be represented by $B = b_{m-1}b_{m-2}\cdots b_1b_0$,

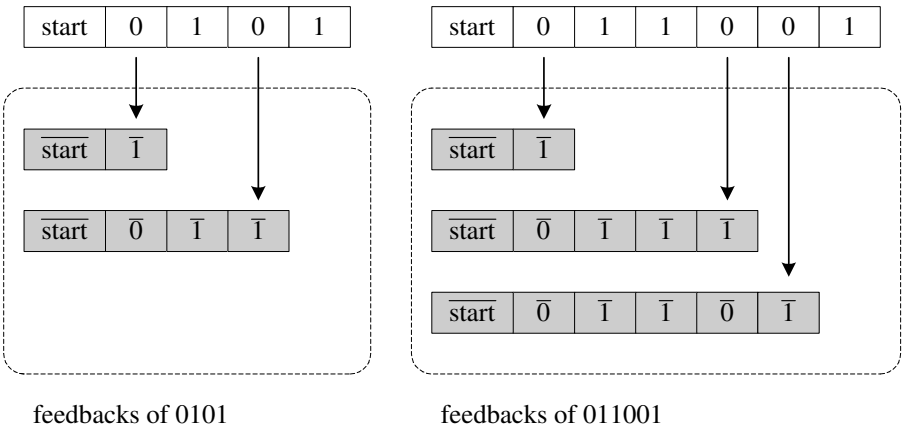

FIGURE 2. Examples of feedbacks for $B = 0101$ and $B' = 011001$.

where each b_i is of value either 0 or 1. For each bit with value 0 in a randomly selected number, we generate one sort of feedback sequence by replacing such a bit of value 0 with value 1. The set of feedbacks for B, denoted as $F(B)$, consists of the following elements: $c(b_{m-1} \cdots b_{i+1}1)$ if $b_i = 0$, where $c(b_{m-1} \cdots b_{i+1}1)$ represents the complementary DNA strand of the parenthesized number. For example, as shown in Figure 2, the feedbacks for $B = 0101$ and $B' = 011001$ are $F(B) = \{c(1), c(011)\}$ and $F(B') = \{c(1), c(0111), c(01101)\}$.

In each iteration, we select ω portion of DNA strands and use them for feedback generation. After pouring the feedbacks into the original tube, the feedbacks will simultaneously anneal to the DNA strands with values greater than the randomly selected values. Such processes of randomly picking numbers, generating their feedbacks, and further pouring them to the original pool can be viewed as choosing nodes and broadcasting their values. However, there are two differences between BCM and the DNA computing system. In BCM, only one value is allowed to be broadcast via a common channel or bus in a computing iteration, whereas more than one value in a DNA computer is transmitted simultaneously in a computing iteration. Moreover, in the way of communication in BCM, all other nodes receive the same message (value) at a time; while in DNA computing, each node receives a unique message (value) from another node, though different messages may represent the same value.

Figure 3 provides a global view of our method on finding the minimum among 16 binary numbers. We first describe our concept under the perfectly ideal environment that no partial or wrong hybridization occurs. Suppose that the numbers 0010, 0011, and 1001 are randomly chosen and they produce feedbacks $\{c(1), c(01), c(0011)\}$, $\{c(1), c(01)\}$ and $\{c(11), c(101)\}$, respectively. After the feedbacks are poured back into the pool, the numbers larger than 0010, 0011, or 1001 are attacked and removed. In the end of this iteration, the updated pool contains the initial amount of 0010, 0001, and 0000 and the reduced amount of the attacked numbers. Such procedure repeats until the minimum is determined.

Since our method is iterative and the wrong answers are eliminated gradually, we can tolerate some chemical reaction errors. Although the feedbacks may attack parts of strands representing smaller numbers, we can get the minimum eventually. We shall present our analysis for tolerating errors in Section 4.

3. **Methods.**

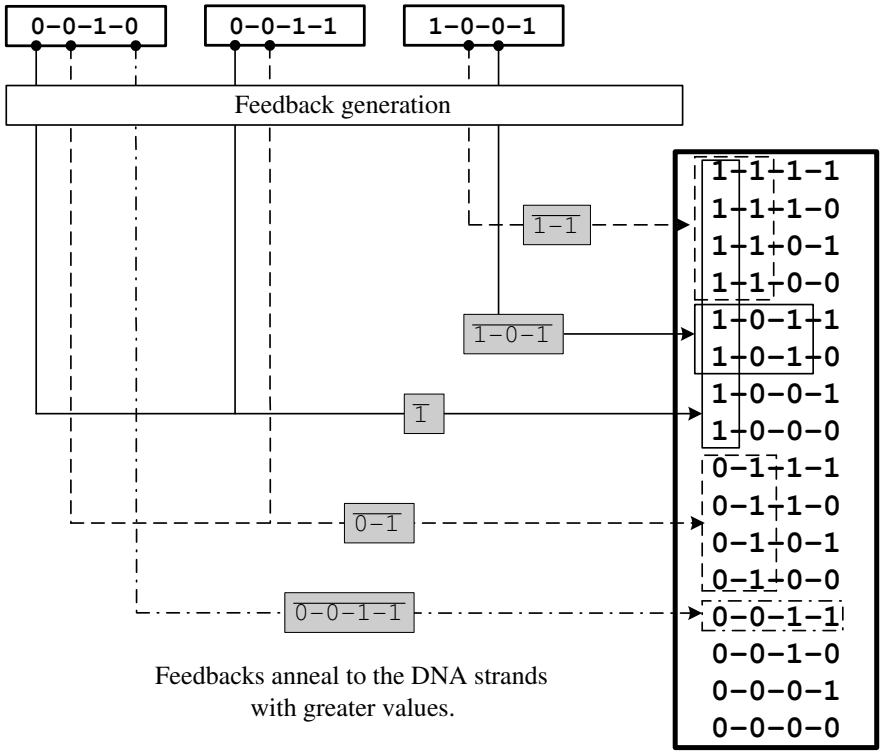

FIGURE 3. A global view of minimum finding.

3.1. **The DNA encoding forms and feedback generation.** As shown in Figure 1, value 0 at bit i is composed of α_i and β_i whereas value 1 is composed of α_i and γ_i. Meanwhile, there is one restriction site on value 0, but no restriction site on value 1. Accordingly, the encoding forms of 0 and 1 are different from each other and the encoding forms of different bit positions with the same value are varied with bit positions.

3.2. **Algorithm: Generating feedbacks.**

Step 2.1:: Divide the randomly picked DNA strands into m tubes (t_{m-1}, t_{m-2}, $\cdots$, t_0), where tube t_i will generate feedbacks of length $m-i$, $0 \leq i \leq m-1$.

Step 2.2:: Pour sequences $\overline{\alpha_i\beta_i}$ into tubes t_i, where $0 \leq i \leq m-1$. For an input number, sequences $\overline{\alpha_i\beta_i}$ will anneal to $\alpha_i\beta_i$ at bit position i with value 0. Numbers with value 1 at position i are not affected.

Step 2.3:: In each tube t_i, add the corresponding restriction enzymes to digest the partially double-stranded DNA segments to cut α_i and β_i of bit position i with value 0.

Step 2.4:: In each tube t_i, keep the bit string starting from bit $m-1$ to bit i of the DNA strand by applying the gel electrophoresis to tell the varied molecular weight.

Step 2.5:: Pour $\overline{\alpha_i\gamma_i}$ into tubes t_i to transform the value of bit i from 0 to 1.

The amount of feedbacks can be amplified by using PCR to reduce the number of iterations. That is, we can amplify the ω portion before performing Algorithm Generating Feedbacks, amplify the feedbacks after Algorithm Generating Feedbacks, or both.

Figure 4 brings an example of generating feedbacks $c(1)$, $c(01)$, $c(11)$, $c(101)$, and $c(0011)$ from numbers 0010, 0011, and 1001. We design each of the m bits as 20 bases long, 10 bases for α and β/γ, with a site in length of six bases recognizable by restriction enzyme in the middle. The sequence length for 'start' is 50 bases longer than the part encoded with the m bits, so that the gel electrophoresis can separate the desired and

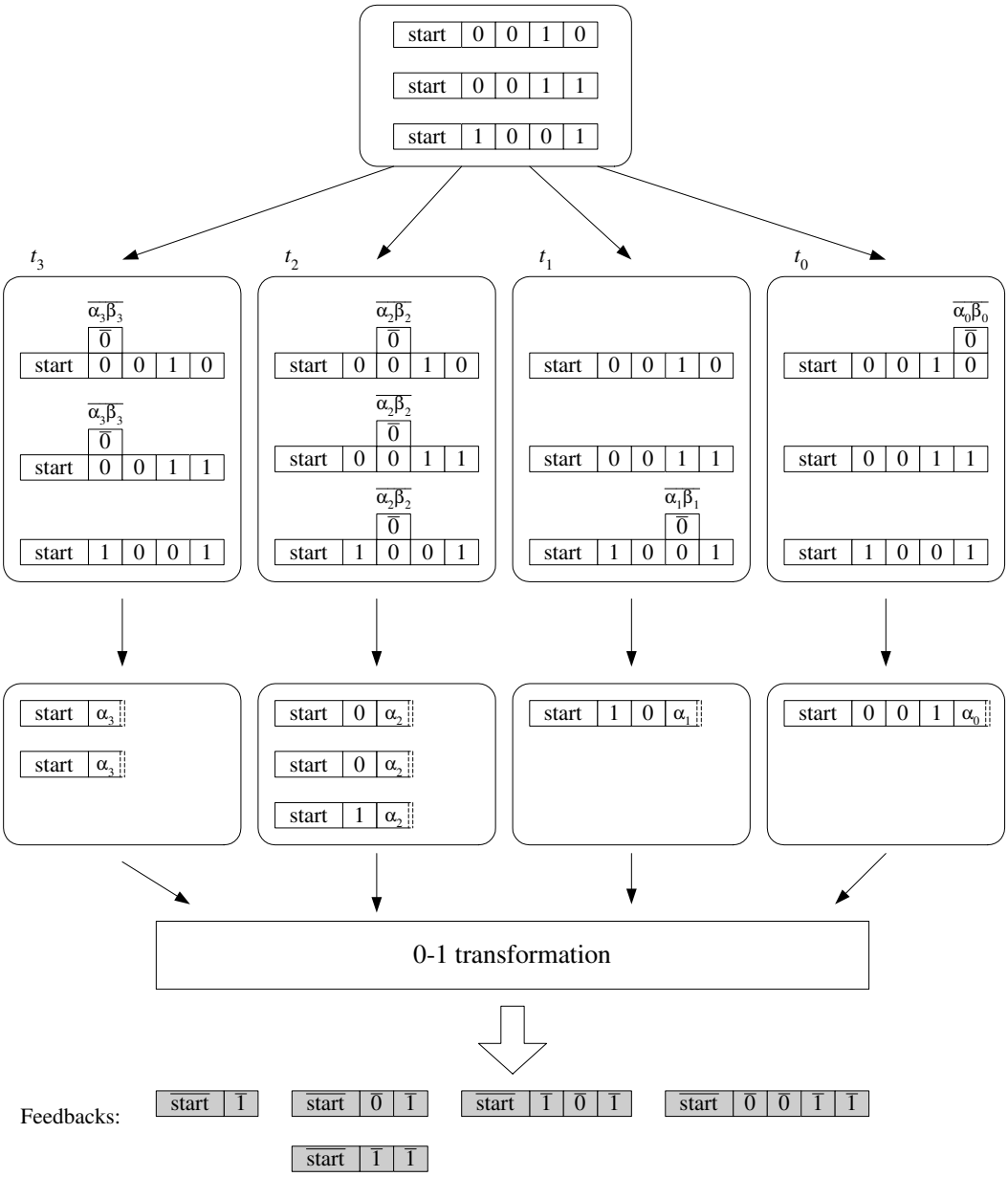

FIGURE 4. The flow chart of the feedback generation for 0010, 0011, and 1001.

discarded strands in Step 2.4. For example, the DNA sequence for a 4-bit number 'start'-0010 has 210 bases (20*4 = 80 for 4 bits and 50+80 = 130 for 'start'). Accordingly, the 'start'-α_1, 'start'-0-α_2, 'start'-1-0-0-α_0 are in length of 140, 160, and 200 bases respectively which can be distinguished from the discarded fragments in length of 70, 50, and 10 bases.

Figure 5 illustrates the detailed transformation in Step 2.5, where the detail of transforming 00 to 01 is given. The sequence 'start'-0-α_2 anneals to the added $\overline{\alpha_2\gamma_2}$ and further, through primer extension, γ_2 will grow adjacent to α_2 so that the original 00 becomes 01. Next, the temperature is raised to peel off $\overline{\alpha_2\gamma_2}$ and sequences 'start'-0-1 are collected by gel electrophoresis separation. With primer extension, double-stranded DNA 'start'-0-1 is formed and the temperature is raised again to denature the double-stranded sequences and this time sequences '$\overline{\text{start}}$'-$\overline{0}$-$\overline{1}$ are collected as feedbacks to capture numbers such as 0100, 0101, 0110, and 0111 later.

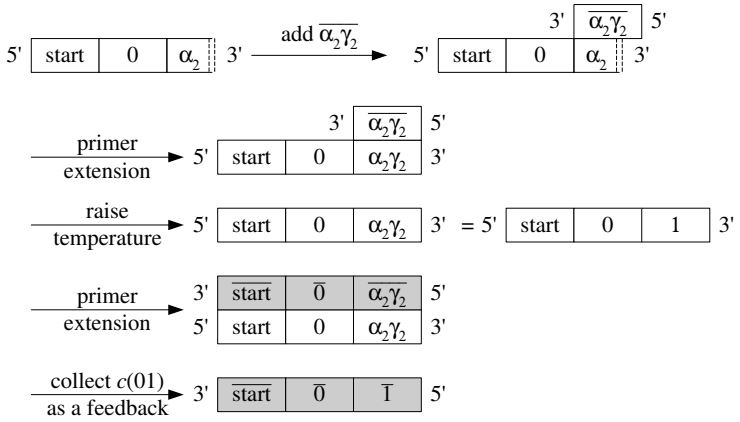

FIGURE 5. The flow chart of the 0-1 transformation from 00 to 01.

3.3. **Data pool update.** The process of updating data pool is illustrated in Figure 6. On the top, it shows the pool prior to update. After adding the feedbacks, one part of the pool is double-stranded and the other part is single-stranded, which is to be collected. The separation process is achieved with probe recognizing the unannealed 'start' fragment. As the mixture passes a column with probes, the single-stranded sequences will be retained while the double ones will drain out and be discarded.

4. **Analysis.** Figure 7 illustrates a relationship diagram of input numbers and their feedbacks in a 4-bit number system consisting of 16 distinct input numbers, where the sharp rectangles and rounded rectangles represent input numbers and the derived feedbacks, respectively. The solid arrows pointing from a sharp rectangle to several rounded rectangle connect a number and its corresponding feedbacks; the dashed arrows pointing from a rounded rectangle to several sharp rectangle link a feedback and the possible numbers eliminated by it. Numbers with more bits of value 0 generate more feedbacks whereas numbers with more bits of value 1 may be killed by more kinds of feedbacks. Let $B^k = b^k_{m-1}b^k_{m-2}\cdots b^k_1 b^k_0$ be the binary representation of number k. For example, consider the number $k = 0101$. The bold arrows in Figure 7 illustrate that feedbacks $c(0101)$ and $c(01)$ play the roles for eliminating k. Number 0100 will generate $c(0101)$ at bit 0; numbers 0000, 0001, 0010, and 0011 will generate $c(01)$ at bit 2. Note that the feedback $c(01)$ also eliminates 0100, 0110, and 0111.

For simplifying the analysis of the complex m-bit number system, we assume that all numbers within 0 and $2^m - 1$ are the input data. Furthermore, the concept of corresponding killers is proposed. The *corresponding killer* of number k at bit j (of value 1) is defined to be $(k - 2^j)$, which is obtained by changing bit j of k from value 1 to value 0. It can generate the feedback to eliminate k. As shown in Figure 7, the dashed arrows mean that for feedback $c(01)$, numbers 0000, 0001, 0010 and 0011 are the corresponding killers at bit 2 of 0100, 0101, 0110 and 0111, respectively. For example, we take $(k - 2^j) = (k - 2^2)$ $= 0101 - 0100 = 0001$ as the corresponding killer at bit 2 for $k = 0101$. Note that in the real chemical reaction, $k = 0101$ can be killed by any number generating feedbacks $c(01)$ or $c(0101)$, which may be 0100, 0101, 0110, 0111 and 0100. The concept of corresponding killer makes the complex system more clear and the analysis easier.

In Algorithm Minimum Finding, assuming that there are n input numbers with m binary bits, and we randomly choose ω portion from the data pool. Suppose we can amplify the feedbacks with factor σ using PCR before or after Algorithm Generating Feedbacks.

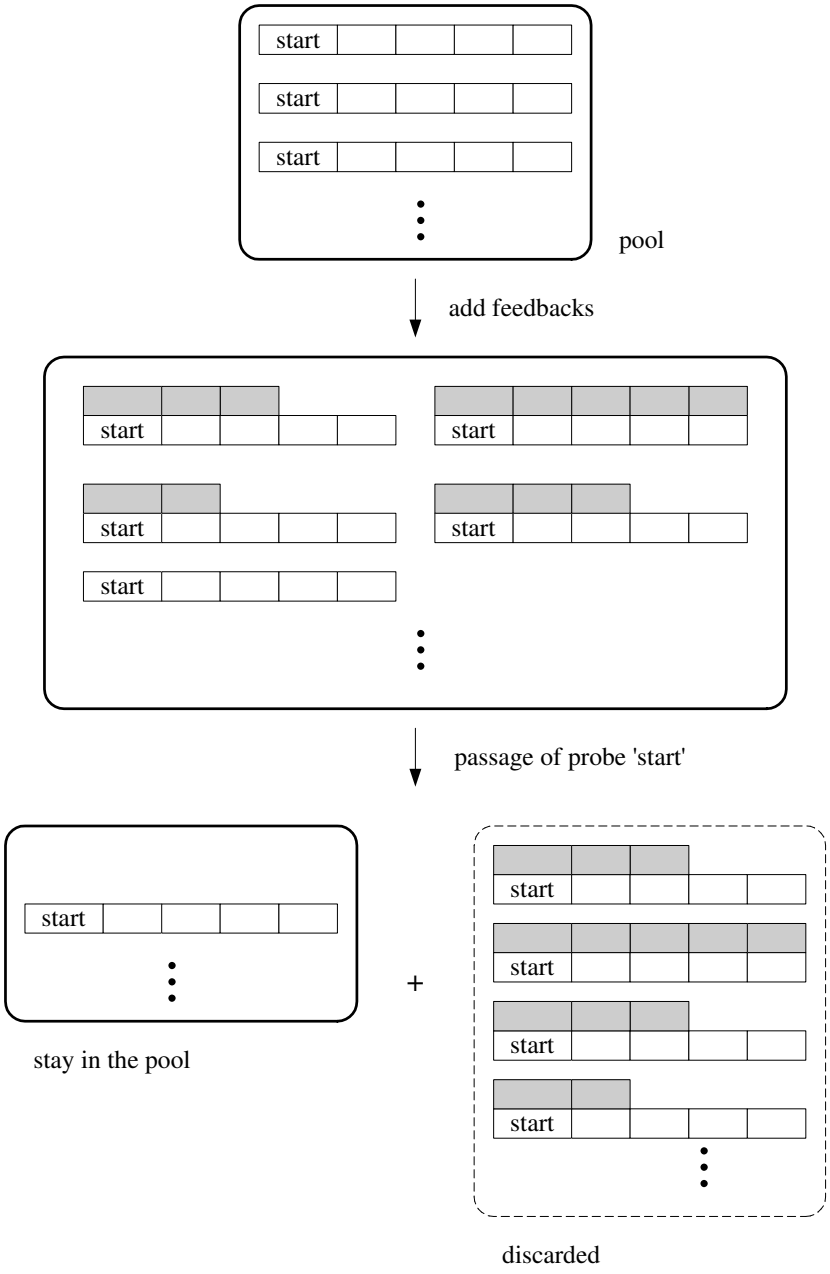

FIGURE 6. Flow chart of one computing iteration.

We define $N_k(i)$ as the remaining amount of number k after the ith iteration. The symbol $N'_k(i)$ represents the amount of corresponding killers (feedbacks) for eliminating k.

$$N'_k(i) = \sum_{j=0}^{m-1} [N_{(k-2^j)}(i-1) \cdot b_j^k] \tag{1}$$

$$N_k(i) = \max\{(1-\omega)N_k(i-1) - (\frac{\omega\sigma}{m})N'_k(i), 0\} \tag{2}$$

In Equation 2, $(1-\omega)N_k(i-1)$ represents the remaining amount of k after pouring ω portion in iteration $i-1$. The ω portion of the corresponding killer to generate feedbacks is $\omega N'_k(i)$, and note that we can amplify the feedbacks with factor σ. In addition, the

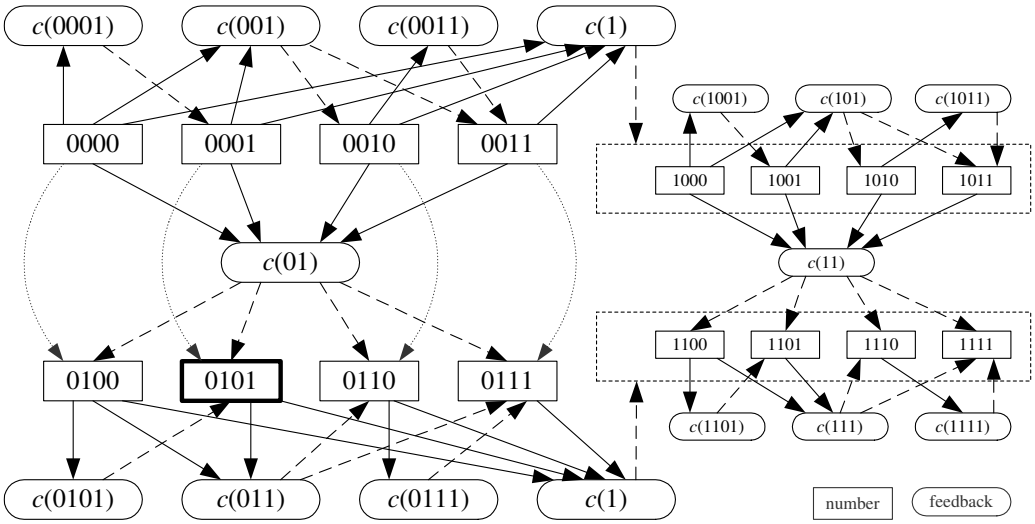

FIGURE 7. The relationship diagram of input numbers and their feedbacks in a 4-bit number (16 numbers) system.

feedbacks are divided into m tubes evenly. So we multiply the factor $(\omega\sigma/m)$ to $N'_k(i)$. One may note that $N_k(i)$, the amount of k, should not be negative.

In deed, $N_k(i)$ depends on other numbers with identical prefix sequence because they share the same kind of feedbacks. For example, feedback $\overline{\text{'start'-0-1}}$ will capture the numbers with prefix 01 such as 0100, 0101, 0110, and 0111. Hence, the feedback sharing property can raise the portion of the smallest number at accelerated pace.

Some errors, such as operating waste and miss, may occur during a biochemical experiment. For more precise analysis, these experimental errors should be discussed. In Algorithm Minimum Finding, Step 2 invokes Algorithm Generating Feedbacks in Section 3.2; Steps 3 and 4 are accomplished by the data pool update in Section 3.3. Let $p(Ef)$ and $p(Eu)$ denote the error probabilities of generating feedbacks and updating data pools, respectively. In Algorithm Generating Feedbacks, there are five steps. Let $p(Ef_j)$, $1 \leq j \leq 5$, denote the error probability for each step in Algorithm Generating Feedbacks. Let $p(Eu_3)$ and $p(Eu_4)$ denote the error probabilities of Steps 3 and 4 in Algorithm Minimum Finding, respectively. Note that the possible errors of partial or wrong hybridization mentioned in Section 2.3 are counted in $p(Eu_3)$. The generating feedback error probability $p(Ef)$ can be formulated as follows.

$$p(Ef) = 1 - \prod_{j=1}^{5}(1 - p(Ef_j)) \tag{3}$$

If the experimental error probabilities $p(Ef)$, $p(Eu_3)$ and $p(Eu_4)$ are considered, $N_k(i)$ can be modified as follows.

$$N_k(i) = [1 - p(Eu_4)] \cdot \max\{(1-\omega)N_k(i-1) - [1 - p(Eu_3)][1 - p(Ef)](\frac{\omega\sigma}{m})N'_k(i), 0\} \tag{4}$$

For convenience, let $N_s(i)$ denote the amount of the smallest number. The portion of the smallest number after i iterations can be formulated as $P(i)$,

$$P(i) = \frac{|\text{the small number}|}{|\text{total numbers}|} = \frac{N_s(i)}{\sum_{k=0}^{n-1} N_k(i)}. \tag{5}$$

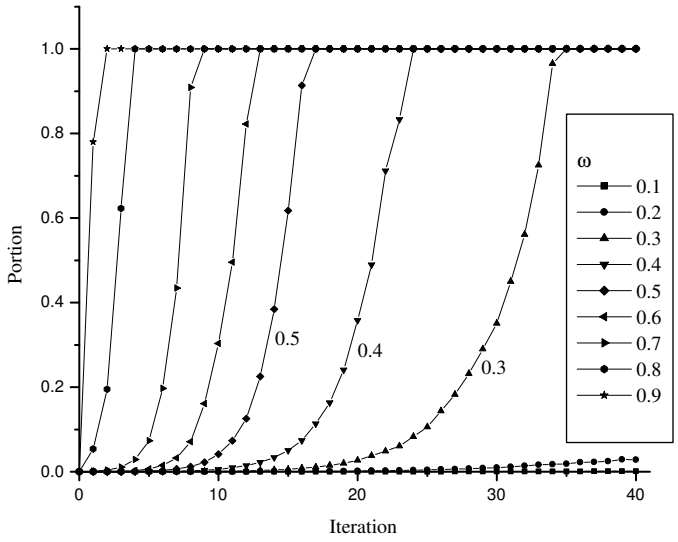

FIGURE 8. The portion diagram of the minimum in the set, where $n = 65536$, $\sigma = 1$, and ω varies from 0.1 to 0.9.

Based on these formulas, one can predict the number of iterations or the amplifying ratio σ in order to achieve a specific portion of the smallest number. In addition, variables i and σ can help us to control and determine the amount of input sequences for each input number and required iterations for amplifying feedbacks at the initial stage, respectively. It is clear that if ω or σ increases, the number of required iterations would decrease.

For illustrating Algorithm Minimum Finding, some computer simulations are performed. In these simulations, there are 65536 16-bit numbers with 1000 clones for each number, and various parameter values for ω, σ, and error are considered. Figures 8, 9 and 10 show the portion diagrams of the minimum for ω ($0.1 \leq \omega \leq 0.9$), σ ($1 \leq \sigma \leq 128$), and individual error probabilities ($p(Ef_j)$, $1 \leq j \leq 5$, $p(Eu_3)$ and $p(Eu_4)$) from 0 to 0.2, respectively. In Figure 8, when $\omega = 0.9$, the portions of the minimum number are 0.78 and 1 after one and two iterations, respectively. In Figure 9, when $\sigma = 128$, it requires three iterations so that almost all of the strands are the minimum; the portion of the minimum is 0.906 after one iteration. Figure 8 and Figure 9 show and conclude that larger ω or σ can make higher portion of the smallest number in fewer iterations. Figure 10 shows that our algorithm can allow some scale of experimental errors. In Figure 10, all small error probabilities, such as 0, 0.01 and 0.02, require only two iterations to make the portion of the minimum reach almost 1. In addition, when the error probability is large, such as 0.2, our algorithm can still work. It is clear that, with higher error probabilities, we require more iterations to get higher density of the small number. Note that if each $p(Ef_j) = 0.2$, it implies $p(Ef) = (1-(1-0.2)^5) = 0.67232$. It means that 67.23% quantity is missing in the step of generating feedbacks. In such experiment situations of high errors, our algorithm can still work. It concludes that our algorithm has the ability of error tolerance.

5. **Concluding Remark.** We propose a randomized minimum finding algorithm with DNA computing in the binary-numbered system. By implementing the feedback scheme with the help of restriction enzymes, the wrong answers in the test tube can be greatly reduced. PCR is used to duplicate the solutions with correct bit size (m bits). The

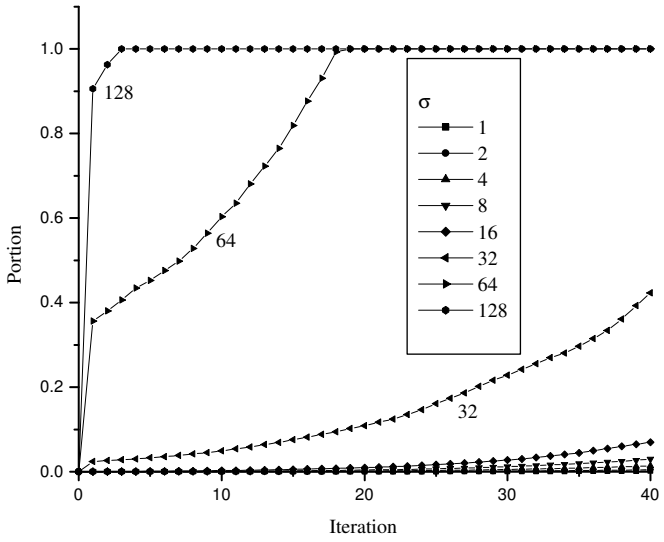

FIGURE 9. The portion diagram of the minimum in the set, where $n = 65536$, $\omega = 0.1$, and σ varies from 1 to 128.

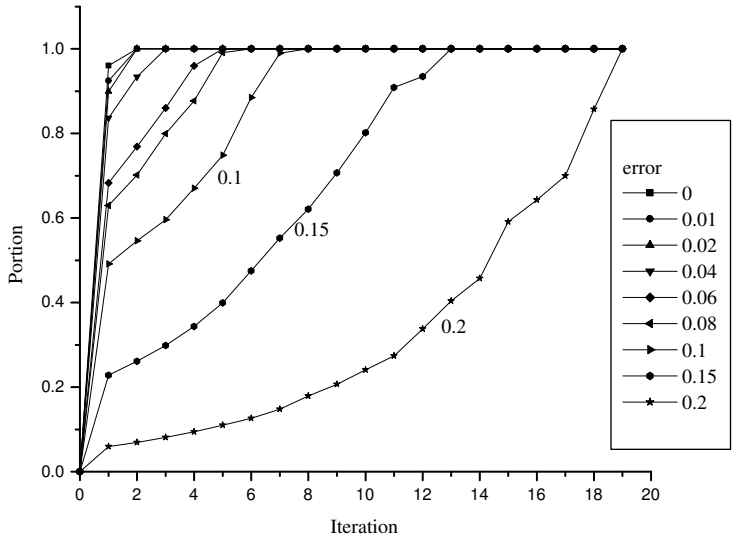

FIGURE 10. The portion diagram of the minimum in the set, where $n = 65536$, $\omega = 0.2$, $\sigma = 64$, with error probabilities ($p(Ef_j)$, $1 \leq j \leq 5$, $p(Eu_3)$, and $p(Eu_4)$) are 0, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.15, and 0.2, respectively.

solution space is divided into two parts, namely, correct and incorrect. Accordingly, we believe such an idea can be widely applied to decrease the effect of possible chemical reaction errors during the biochemical experiments. In other words, our method has the ability to tolerate chemical reaction errors. Another unique in this work is instead of using conventional length-dependent sequences to represent numbers of different values, a

binary coding on DNA strands is invented to closely mimic practical computer engineering situations, especially for solving a numerical problem covering a wide range of values.

On current silicon-based computers, a sequential minimum finding algorithm requires $O(n)$ time to find out the minimum among n input numbers. In this paper, our minimum finding algorithm uses the broadcasting concept on BCM. The time complexity is $O(\log n)$ on BCM whereas our DNA minimum finding algorithm requires only one iteration if quantity of feedbacks is very large and uniformly distributed, and the chemical experiments are done in the very ideal environment that there is no chemical reaction errors. The actual number of iterations required depends on the reaction error rate.

Besides, our idea can also be applied to solving hard problems in DNA computing style, such as the traveling salesperson problem or the 0/1 knapsack problem. Among 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. Therefore, by increasing the quantity of the optimal solutions, the designed experiment will be performed in a more efficient way. Our algorithm has the ability to tolerate errors and thus, it can serve as a novel strategy for optimization in DNA computing. The practicability and efficiency of DNA computing algorithms are determined by bio-chemical techniques, such as PCR and manipulations of tubes. New techniques and improvements of bio-chemistry will make DNA computing more feasible and efficient. 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 can be improved. Some approximation algorithms may be designed based on these strategies. Therefore, hard problems may be solved more efficiently with DNA computing.

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

REFERENCES

[1] R. B. Abu Bakar, J. Watada, and W. Pedrycz, A proximity approach to DNA based clustering analysis, *International Journal of Innovative Computing, Information and Control*, vol.4, no.5, pp.1203-1212, 2008.

[2] L. M. Adleman, Molecular computation of solutions to combinatorial problems, *Science*, vol.266, no.5187, pp.1021-1024, 1994.

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

[4] 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.

[5] J. I. Capetanakis, Tree algorithms for packet broadcast channels, *IEEE Transactions on Information Theory*, vol.25, no.5, pp.505-515, 1979.

[6] C.-C. Chang, T.-C. Lu, Y.-F. Chang, and C.-T. Lee, Reversible data hiding schemes for deoxyribonucleic acid (DNA) medium, *International Journal of Innovative Computing, Information and Control*, vol.3, no.5, pp.1145-1160, 2007.

[7] R. Dechter and L. Kleinrock, Broadcast communications and distributed algorithms, *IEEE Transactions on Computers*, vol.35, no.3, pp.210-219, 1986.

[8] A. Dove, From bits to bases: Computing with DNA, *Nature Biotechnology*, vol.16, no.9, pp.830-832, 1998.

[9] S. P. A. Fodor, Massively parallel genomics, *Science*, vol.277, no.5324, pp.393-395, 1997.

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

[11] 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.

[12] M. Hagiya, From molecular computing to molecular programming, *Proc. of the 6th International Workshop on DNA-Based Computers, LNCS*, vol.2054, pp.89-102, 2001.
[13] T. Hinze and M. Sturm, A universal functional approach to DNA computing and its experimental practicability, *Proc. of the 6th DIMACS Workshop on DNA Based Computers, The University of Leiden*, Leiden, The Netherlands, pp.257-266, 2000.
[14] I. Kim, D. J.-F. Jeng, and J. Watada, Redesigning subgroups in a personnel network based on DNA computing, *International Journal of Innovative Computing, Information and Control*, vol.2, no.4, pp.885-896, 2006.
[15] S. P. Levitan and C. C. Foster, Finding an extremum in a network, *Proc. of the 9th Annual Symposium on Computer Architecture*, Austin, Texas, United States, pp.321-325, 1982.
[16] C. H. Lin, H. P. Cheng, C. N. Yang, and C. B. Yang, Solving satisfiability problems using a novel microarray-based DNA computer, *BioSystems*, vol.90, no.1, pp.242-252, 2007.
[17] R. J. Lipton, DNA solution of hard computational problems, *Science*, vol.268, no.5210, pp.542-545, 1995.
[18] D. L. Robertson and F. G. Joyce, Selection in vitro of an RNA enzyme that specifically cleaves single-stranded DNA, *Nature*, vol.344, no.6265, pp.467-468, 1990.
[19] S. H. Shiau and C. B. Yang, A fast maximum finding algorithm on broadcast communication, *Information Processing Letters*, vol.60, pp.81-89, 1996.
[20] S. H. Shiau and C. B. Yang, A fast initialization algorithm for single-hop wireless networks, *IEICE Transactions on Communications*, vol.E88-B, no.11, pp.4285-4292, 2005.
[21] W. P. C. Stemmer, Rapid evolution of a protein by DNA shuffling, *Nature*, vol.370, no.6488, pp.389-391, 1994.
[22] K. Tanaka, A. Okamoto, and I. Saito, Public-key system using DNA as a one-way function for key distribution, *Biosystems*, vol.81, no.1, pp.25-29, 2005.
[23] H. Y. Wang, C. B. Yang, K. S. Huang, and Y. L. Shiue, The design of sorters based on DNA for bio-computers, *Proc. of the International Computer Symposium, Workshop on Algorithms and Computational Molecular Biology*, Hualien, Taiwan, 2002.
[24] C. B. Yang, R. C. T. Lee, and W. T. Chen, Parallel graph algorithms based upon broadcast communications, *IEEE Transactions on Computers*, vol.39, no.12, pp.1468-1472, 1990.
[25] C. N. Yang and C. B. Yang, A DNA solution of SAT problem by a modified sticker model, *BioSystems*, vol.81, no.1, pp.1-9, 2005.