# Error-Tolerant Minimum Finding with DNA Computing

Chia-Ning Yang
Department of Medical Imaging and Radiological Sciences,
I-Shou University, Kaohsiung 840, Taiwan

Kuo-Si Huang, Chang-Biau Yang[*] and Chie-Yao Hsu
Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung 804, Taiwan

**Abstract**

Nearly 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 work, an algorithm for solving the minimum finding problem is proposed in theory with a randomized scheme -- the broadcasting scheme on the broadcast communication model. In our algorithm, chemistry experiment errors have been taken into consideration. Thus, our algorithm can tolerate some chemistry reaction and experiment errors. To demonstrate the feasibility of the algorithm and the designed experiment, several tasks of simulation were also performed and analyzed.

***Key words:*** 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 [1,6,9-11,13,20]. Compared to the traditional silicon-based computing system, there are two advantages in DNA computers -- high capacity of data storage and massive parallelism implemented in coding. For instance, there are almost $6\times10^{17}$ DNA molecules in 1 $\mu$mol and under the circumstance of one bit being encoded with one nucleotide, 1 $\mu$mol DNA molecules hold $6\times10^{17}$ bits of information, which is much more than the most massive scale $10^{12}$ bits (Tera bits) of the storage in a regular computer at current stage. In DNA computing, different data can be encoded into DNA strands. DNA strands can be processed with the biomolecular operations such as primer extension, PCR, restriction enzyme digestion, and gel electrophoresis [3,8,14,17,18]. 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 parallel logical or arithmetic processing in the traditional computers. Moreover, because the interactions take place in the tube simultaneously, we can regard DNA computing 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) [2,7].

Generally speaking, DNA computing often takes the strategy of brute force search in solving problems. That is, a library of all possible answers are considered at very beginning and a series of biomolecular laboratory techniques are systematically applied to eliminate the wrong answers and collect the right ones for answer readout.

[*] All correspondence should be addressed to Chang-Biau Yang
Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan 80424 (E-MAIL: cbyang@cse.nsysu.edu.tw)

In this work, a novel method based on DNA computing is developed to solve the minimum finding problem which is very fundamental to computer science and plays a basic role in almost every computer software, from the easiest to the very complicate. Even in solving NP-hard problems, such as the traveling salesperson problem, minimum finding has to be invoked in many heuristics. The broadcast communication model (BCM) is a simple computation model. For example, the famous Ethernet is its implementation and a number of researchers have designed algorithms for solving problems on BCM [4, 5,16,19]. The minimum (or maximum) finding has a good implementation on BCM [12,15]. Here, we can apply the concept of broadcasting on BCM for solving the minimum finding problem with DNA computing. Meanwhile, we also performed a series of computer simulation of the proposed algorithm to demonstrate and discuss 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 [12,15], which consists of some processors sharing one common channel for communications. In this model, each processor can communicate with others only through the shared channel. Whenever a processor broadcast messages, any other processor can hear the broadcast messages via the shared channel. If more than one processor wants to broadcast messages simultaneously, a broadcast conflict occurs. The main idea of our algorithm 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 rounds whereas the nodes with values less than *x* hold the chance to broadcast their values in the latter rounds. In the next round, 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* rounds to find out the minimum in the worst case, where *n* denotes the number of nodes in BCM. In average, $O(\log n)$ rounds are required [12,15].

### 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 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 as bit 0 all the way to the left most one 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 the first bit '1' is different from the third bit '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 process is divided into several rounds and in each round 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 rounds, 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 process.

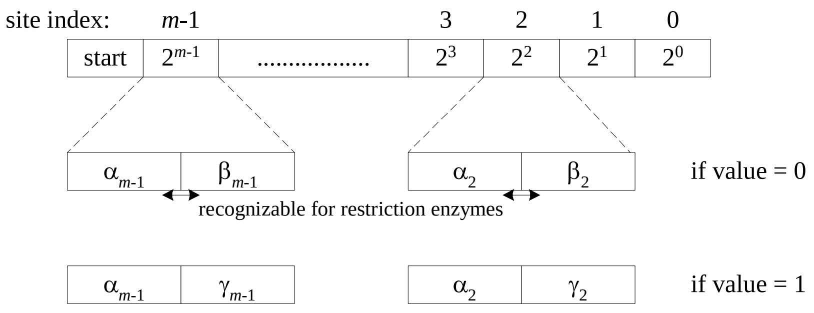

**Figure 1** The assignment of sequence. Note $\alpha$, $\beta$, and $\gamma$ are varied with site $i$.

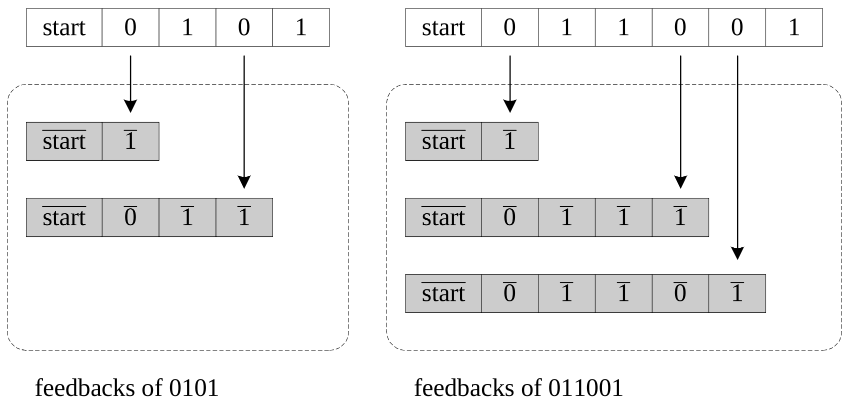

**Figure 2** Examples of feedbacks for 0101 and 011001.

## 2.3 Algorithm: Minimum Finding

Input: A set $A$ of $n$ binary numbers encoded by DNA strands.
Output: The minimum number in set $A$.
Step 1: From solution pool $A$ randomly choose $\omega$ 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 $B = b_{m-1}b_{m-2}\ldots b_1b_0$, 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 $F(B)$, consist of the following elements: $c(b_{m-1}\ldots b_{i+1}1)$ if $b_i = 0$, where $c(b_{m-1}\ldots b_{i+1}1)$ 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 round, we select a part 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 process 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 a 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 are transmitted simultaneously in a computing iteration. Moreover, the way of communication in BCM is that 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 the meanings of some of messages are the same (same values).

Figure 3 provides a global view of our method on finding the minimum among 16 binary numbers. 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 0001, 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 reduced amount of the attacked numbers for further random selection and broadcast until the minimum is determined.

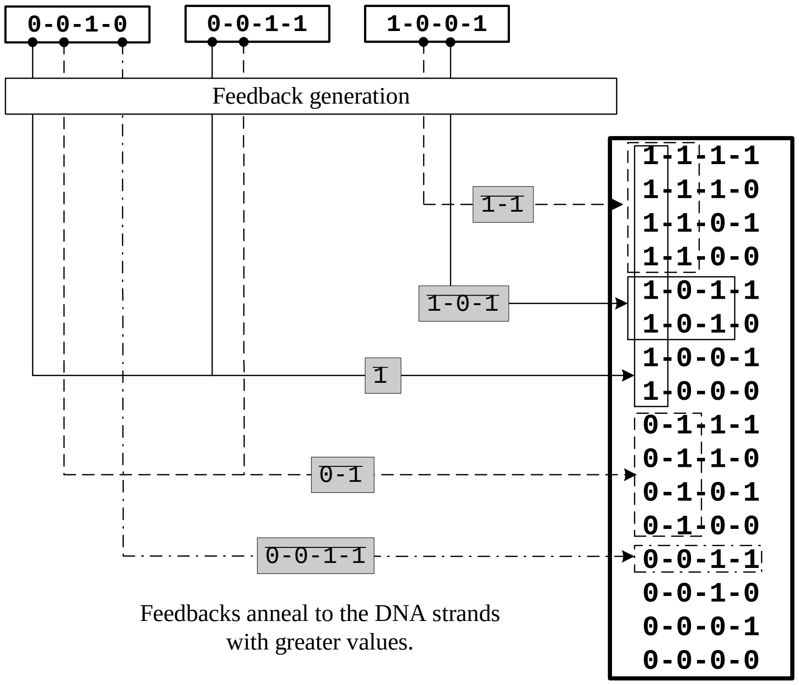

**Figure 3** A global view of minimum finding.

## 3. Methods

### 3.1 The DNA encoding forms and feedback generation

As shown in Figure 1, value 0 at bit *i* is composed of $\alpha_i$ and $\beta_i$ whereas value '1' is composed of $\alpha_i$ and $\gamma_i$. 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 position.

### 3.2 Algorithm: Generating Feedbacks

Step 2.1: Divide the randomly picked DNA strands into *m* tubes ($t_{m-1}$, $t_{m-2}$, …, $t_0$) where feedbacks in different lengths will be generated in different tubes.

Step 2.2: Pour sequences $\overline{\alpha_i\beta_i}$ into tubes $t_i$, $0 \le i \le m - 1$. Sequence $\overline{\alpha_i\beta_i}$ will anneal to $\alpha_i\beta_i$ at bit position *i* with value '0' of an input number. 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 $\alpha_i$ and $\beta_i$ of bit position *i* with value 0.

Step 2.4: In each tube $t_i$, keep the bit string starting from the most significant bit 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 tube $t_i$ to transform the value of bit *i* from '0' to '1'.

For reducing the number of iterations, the amount of feedbacks can be amplified by using PCR. We can amplify the $\omega$ portion before performing Algorithm Generating Feedbacks, amplify the feedbacks after Algorithm Generating Feedbacks, or both.

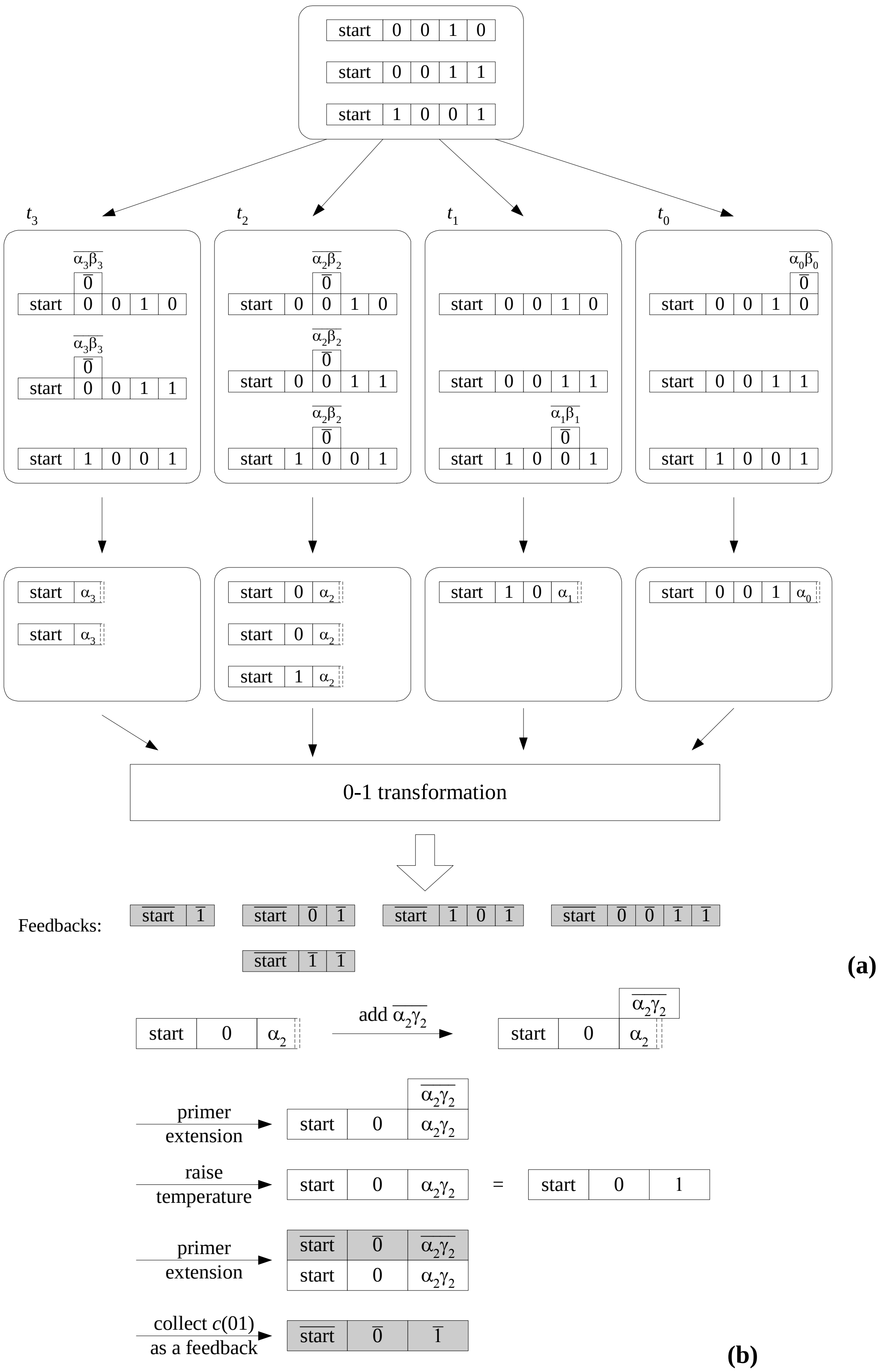
start 0 0 1 0
start 0 0 1 1
start 1 0 0 1
$t_3$
$t_2$
$t_1$
$t_0$
$\overline{\alpha_3\beta_3}$
$\overline{\alpha_2\beta_2}$
$\overline{\alpha_1\beta_1}$
$\overline{\alpha_0\beta_0}$
0-1 transformation
Feedbacks:
(a)
add $\overline{\alpha_2\gamma_2}$
primer extension
raise temperature
collect $c$(01) as a feedback
(b)

**Figure 4** Flow charts of **(a)** feedback generation for 0010, 0011, and 1001 and **(b)** 0-1 transformation from 00 to 01.

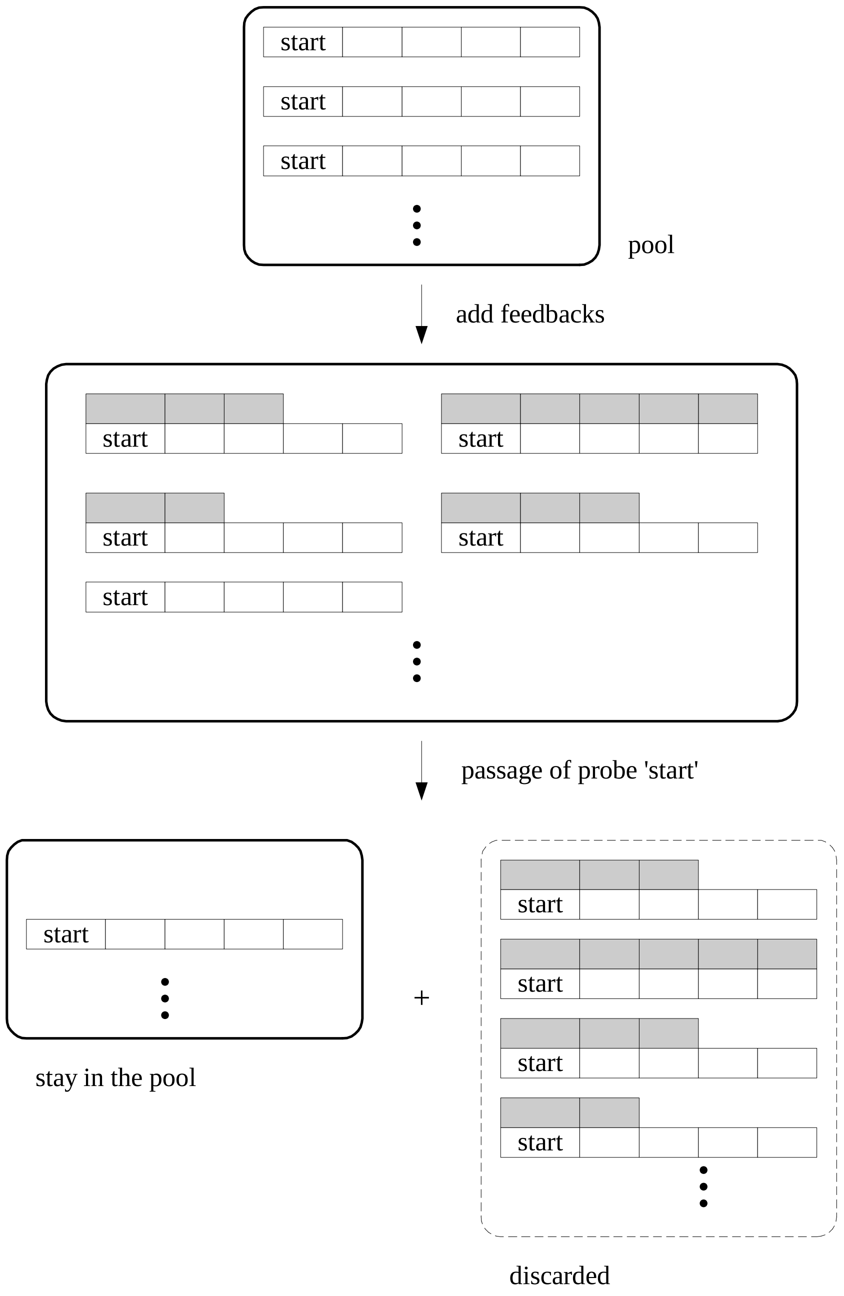

**Figure 5** Flow chart of one computing iteration.

Figure 4(a) brings an example of generating feedbacks 1, 01, 11, 101, and 0011 for 0010, 0011, and 1001. In Figure 4(b), the detail of transforming 00 to 01 is given. The sequence start-0-$\alpha_2$ anneals to the added $\overline{\alpha_2\gamma_2}$ and further, through primer extension, $\gamma_2$ will grow in adjacent to $\alpha_2$ so that the original 00 becomes 01. Next, the temperature is raised to peer off $\overline{\alpha_2\gamma_2}$ and sequences start-0-1 are collected. 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 $\overline{\text{start-0-1}}$ are collected as feedbacks to capture numbers such as 0100, 0101, 0110, and 0111 later.

**3.3 Data pool update**

The process of updating data pool is illustrated in Figure 5. On the top, it shows the pool prior to update. After adding the feedbacks, part of the pool is double-stranded and another part 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

As mentioned above, there are $n$ values with $m$ binary bits, and we randomly choose $\omega$ portion from the data pool in Algorithm Minimum Finding. Suppose we can amplify the feedbacks with factor $\sigma$ using PCR before or after Algorithm Generating Feedbacks. In the viewpoint of the remaining amount of each number, we define $N_k(i)$ as the remaining amount of value $k$ after the $i$th iteration. Let $B^k = b^k_{m-1}b^k_{m-2}\cdots b^k_1 b^k_0$ be the binary representation of $k$. The symbol $N'_k(i)$ represents the amount of corresponding killers for eliminating $k$.

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

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

We use an example to explain that we do not consider all of the number with common prefix of $k$. For example, consider the number $k$ =0101. 0100 will generate $c$(0101) with bit 0 ($b^k_0$ =1), and 0000, 0001, 0010, 0011 will generate $c$(01) with bit 2($b^k_2$ =1). $N'_k(i)$ considers the only one corresponding feedback number 0100 with bit 0 but not considers all numbers with bit 2. The reason is that $c$(01) is not only annealed to $k$, but also to 01xx. We take $(k - 2^j) = (k - 2^2)$= 0101 – 0100 = 0001 as the corresponding killer with bit 2. In the inequality, $(1 - \omega)N_k(i - 1)$ represents the remaining amount of $k$ after pouring $\omega$ portion in iteration $i - 1$. The $\omega$ portion of each number will be used to generate feedbacks as $N'_k(i)$, and note that we can amplify the feedbacks with factor $\sigma$. In addition, the feedbacks are divided into $m$ tubes evenly. So we multiply the factor $(\omega\sigma/ m)$ to $N'_k(i)$. One may note that the amount of $k$ is not negative.

In deed, $N_k(i)$ is related to 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. If some smaller numbers do not exist, the related feedbacks can still capture other larger ones and they will not be wasted. The feedback sharing property can accelerate to raise the portion of the smallest number.

Some errors, such as partial hybridization, operating waste and miss, may occur during a biological experiment. For more precise analysis, we should discuss these experimental errors. 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 $p(Ef_j)$, $1 \le j \le 5$, denote the error probability for each step. Let $p(Eu_3)$ and $p(Eu_4)$ denote the error probabilities of Steps 3 and 4 in Algorithm Minimum Finding, respectively. The generating feedback error probability $p(Ef)$ can formulated as follows.

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

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

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

For convenience, let $N_0(i)$ denote the amount of the smallest number. The portion of the smallest number after $i$ iterations can be formulated as $P(i)$.

$$P(i) = \frac{|\,\text{the small number}\,|}{|\,\text{total numbers}\,|} = \frac{N_0(i)}{\sum_{k=0}^{n} N_k(i)}$$

Based on these formulas, one can predict the number of iterations or the amplifying ratio $\sigma$ in order to achieve a specific portion of the smallest number. In addition, $i$ and $\sigma$ can help us to control and determine the amount of input sequences for each number and iterations for amplifying feedbacks, respectively. It is clear that if $\omega$ or $\sigma$ is increased, the number of required iterations is decreased.

For illustrating our minimum finding algorithm, we perform some simulations in a silicon-based computer. In these simulations, there are 65536 numbers with 1000 clones for each number, and various parameter values for $\omega$, $\sigma$, and error are taken. Figures 6, 7, and 8 show the portion diagrams for $\omega$ ($0.1 \le \omega \le 0.9$), $\sigma$ ($1 \le \sigma \le 128$), and individual error probabilities ($p(Ef_j)$, $1 \le j \le 5$, $p(Eu_3)$ and $p(Eu_4)$) from 0 to 0.2, respectively. In Figure 6, when $\omega = 0.9$, the portions of the minimum number are 0.78 and 1 after one and two iterations, respectively. In Figure 7, 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 6 and Figure 7 show and conclude that larger $\omega$ or $\sigma$ can make higher portion of the smallest number in fewer iterations. Figure 8 shows that our algorithm can allow some scale of experimental errors. In Figure 8, 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. We note that if each $p(Ef_j)$=0.2, we obtain $p(Ef) = (1 - (1 - 0.2)^5) = 0.67232$. In this situation of high error experiment, our algorithm can still work. It concludes that our algorithm has the ability of error tolerance.

## 5. Concluding Remark

In this paper, we propose a randomized minimum finding algorithm with DNA computing. Our goal is to decrease the quantity of wrong solutions in the test tube. We implement the feedback scheme with the help of restriction enzymes. Then, PCR is used to duplicate the solutions with correct bit size ($m$ bits). This way 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 biochemical experiments.

Besides, it can also be applied to solving many hard problems on DNA computing style, such as 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 error 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 can be improved. Some approximation algorithms will be designed according to these strategies. Therefore, hard problems may be solved more efficiently with DNA computing.

A sequential minimum finding algorithm require $O(n)$ 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 [15] 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.

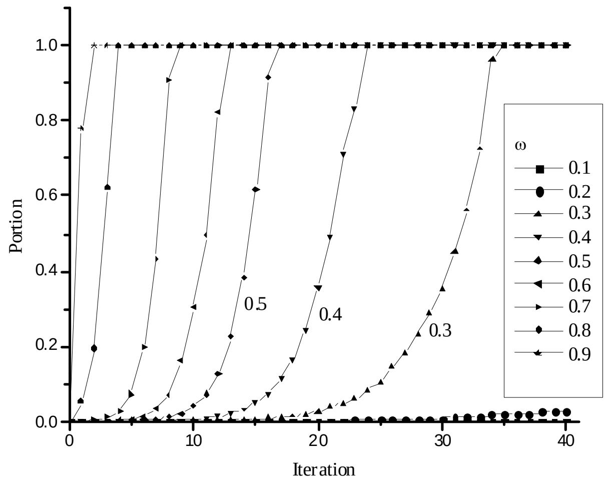

**Figure 6** The portion diagram of the smallest number in the set, where $n$ = 65536 and $\sigma$ = 1, with $\omega$ varied from 0.1 to 0.9.

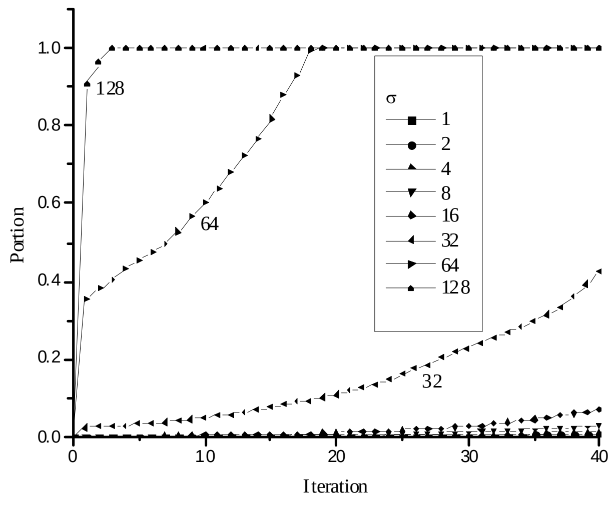

**Figure 7** The portion diagram of the smallest number in the set, where $n$ = 65536 and $\omega$ = 0.1, with $\sigma$ varied from 1 to 128.

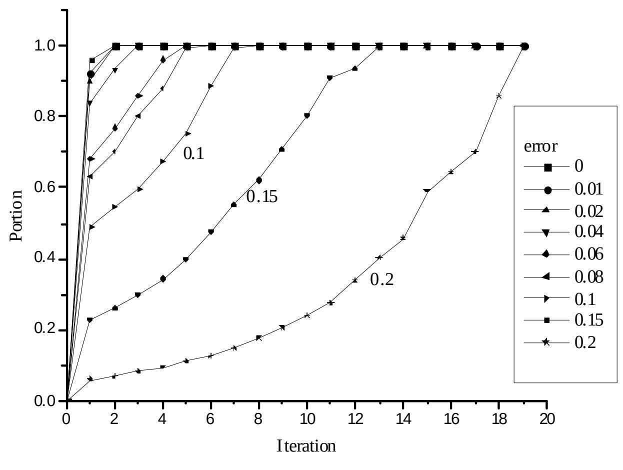

**Figure 8** The portion diagram of the smallest number in the set, where $n$ = 65536, $\omega$ = 0.2, $\sigma$ = 64, with error probabilities ($p(Ef_j)$, $1 \leq j \leq 5$, $p(Eu_3)$, and $p(Eu_4)$) 0, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1, 0.15, and 0.2, respectively.

## Acknowledgement

This work is supported by National Science Council in Taiwan (NSC-91-2213-E-110-022).

## References

[1] Adleman, L.M., 1994. Molecular computation of solutions to combinatorial problems. Science 266, No. 5187, 1021-1024.

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

[3] Akutsu, T., Kuhara, S., Maruyama, O., Miyano, S., 2003. Identification of genetic networks by strategic gene disruptions and gene overexpressions under a Boolean model. Theoretical Computer Science, 298(1), 235-251.

[4] Capetanakis, J. I., 1979. Tree algorithms for packet broadcast channels. IEEE Transactions on Information Theory, 25(5), 505-515.

[5] Dechter, R., Kleinrock, L., 1986. Broadcast communications and distributed algorithms. IEEE Transactions on Computers, 35(3), 210-219.

[6] Dove, A., 1998. From bits to bases: Computing with DNA. Nature Biotechnology 9, 830-832.

[7] Fodor, S.P.A., 1997. Massively parallel genomics. Science 277, 393-395.

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

[9] Gloor, G., Kari, L., Gaasenbeek, M., Yu, S. 1999. Towards a DNA solution to the shortest common superstring problem. International Journal on Artificial Intelligence Tools, 8(4), 385-400.

[10] Hagiya, M. 2001. From molecular computing to molecular programming. DNA Computing, 6th International Workshop on DNA-Based Computers, DNA 2000, Lecture Notes in Computer Science, 2054, 89-102.

[11] Hinze, T., Sturm, M., 2000. 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, 257-266.

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

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

[14] Robertson, D.L., Joyce, F.G., 1990. Selection in vitro of an RNA enzyme that specifically cleaves single-stranded DNA. Nature, 344, 467-468.

[15] Shiau S. H., Yang, C. B., 1996. A fast maximum finding algorithm on broadcast communication," Information Processing Letters, 60, 81-89.

[16] Shiau S. H., Yang, C. B., 2005. A Fast Initialization Algorithm for Single-Hop Wireless Networks. *IEICE Transactions on Communications*, Vol. E88-B, No.11, pp. 4285-4292.

[17] Stemmer, W.P.C., 1994. Rapid evolution of a protein by DNA shuffing, Nature 370, 6488, 389-391.

[18] Wang, H. Y., Yang, C. B., Huang, K. S., Shiue, Y. L., 2002. 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.

[19] Yang, C. B., Lee, R. C. T., Chen, W. T., Dec. 1990. Parallel graph algorithms based upon broadcast communications. IEEE Transactions on Computers, 39 (12), 1468-1472.

[20] Yang, C. N., Yang, C. B., July 2005. A DNA Solution of SAT Problem by a Modified Sticker Model. BioSystems, 81 (1), 1-9.