International Journal of Foundations of Computer Science

PROCEDURES FOR COMPUTING THE MAXIMUM WITH DNA

AKIHIRO FUJIWARA SATOSHI KAMIO AKIKO TAKEHARA

Department of Computer Science and Electronics,
Kyushu Institute of Technology,
680-4 Kawazu, Iizuka, Fukuoka 820-8502, JAPAN
fujiwara@cse.kyutech.ac.jp

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

In recent works for high performance computing, computation with DNA strands, that is, DNA computing, has considerable attention as one of non-silicon based computing. In this paper, we propose three procedures for computing the maximum of n binary numbers of m bits, which are represented with $O(mn)$ DNA strands. The first procedure computes the maximum of the binary numbers in $O(m)$ steps using $O(n)$ kinds of DNA strands. The second and third procedures also compute the maximum in $O(\log n)$ and $O(1)$ steps using $O(mn)$ and $O(mn^2)$ kinds of DNA strands, respectively.

Keywords: DNA computing, the maximum

1. Introduction

In recent works for high performance computing, computation with DNA strands, that is, DNA computing, has considerable attention as one of non-silicon based computing. The DNA has two important features, which are Watson-Crick complementarity and massive parallelism. Using the features, we can solve an NP-complete problem, which usually needs exponential time on a silicon based computer, in a polynomial number of steps with DNA strands. As the first experimental work for DNA computing, Adleman presented an idea of solving the Hamiltonian path problem of size n in $O(n)$ steps using DNA strands. There are a number of other works with DNA strands for combinatorial NP-complete problems [2, 3, 11, 13, 20].

However, the procedures for primitive operations, such as logic or arithmetic operations, are needed to apply DNA computing on a wide range of problems. A number of procedures have been proposed for the primitive operations with DNA strands [5, 7, 8, 9, 10, 15]. Guarnieri et al. [8] has proposed the first procedure for addition using DNA strands. The procedure works in $O(n)$ steps using $O(n)$ different DNA strands for an addition of two n-bit binary numbers. Hug et al. [10] has proposed a procedure for logic operations and addition on the DNA chip. The procedure works in $O(1)$ steps using $O(n)$ different DNA strands for the operation

for n-bit binary numbers. Recently, Fujiwara et al. [7] have proposed addressable procedures for the primitive operations. They showed a data structure which represents n binary numbers of m bits using DNA strands, and proposed a procedure which computes logic operations for any pair of the binary numbers in parallel. The procedure works in $O(1)$ steps using $O(mn)$ kinds of DNA strands. They also proposed a procedure for additions of pairs of two binary numbers. The procedure works in $O(1)$ steps using $O(mn)$ different kinds of DNA strands for additions of $O(n)$ pairs of two binary numbers.

In this paper, we consider three procedures for computing the maximum using DNA strands. We assume that an input of the operation is a set of n binary numbers of m bits, which are represented with $O(mn)$ DNA strands. The first procedure computes the maximum of the binary numbers in $O(m)$ steps using $O(m+n)$ kinds of DNA strands. The procedure consists of a repetition of checking on m bit positions. The second procedure computes the maximum in $O(\log n)$ steps using $O(mn)$ kinds of DNA strands. The procedure consists of a repetition of parallel comparisons of two numbers. The third procedure computes the maximum in $O(1)$ steps using $O(mn^2)$ kinds of DNA strands. The procedure mainly consists of $O(n^2)$ parallel comparisons.

This paper is organized as follows. In Section 2, we give the brief description of the model for DNA computing and data structure for binary numbers represented with DNA strands. In Section 3, we show three kinds of procedures for computing the maximum. Section 4 concludes the paper.

2. Preliminaries

2.1. *Computational model for DNA computing*

A number of theoretical or practical computational models have been proposed for DNA computing [2, 9, 10, 11, 15, 16, 17]. A computational model used in this paper is the same model as [7]. The model is a theoretical model among the proposed models, and biological operations are restricted. Therefore, the model allows the algorithm designer to focus on the structural properties of each problem in DNA computing.

We briefly introduce the model in the following. A *single strand* of DNA is defined as a string of symbols over a finite alphabet Σ. We define the alphabet $\Sigma = \{\sigma_0, \sigma_1, \ldots, \sigma_{m-1}, \overline{\sigma}_0, \overline{\sigma}_1, \ldots, \overline{\sigma}_{m-1}\}$, where the symbol $\sigma, \overline{\sigma}_i$ $(0 \leq i \leq m-1)$ are *complementarity* to each other. Two single strands form a *double strand* if and only if the single strands are complementarity to each other. A double strand with $\sigma_i, \overline{\sigma}_i$ is denoted by $\begin{bmatrix} \sigma_i \\ \overline{\sigma}_i \end{bmatrix}$.

The single or double strands are stored in a *test tube*. For example, $T_1 = \{\sigma_0\sigma_1, \overline{\sigma_1\sigma_0}\}$ denotes a test tube in which the single strands $\sigma_0\sigma_1$, $\overline{\sigma_1\sigma_0}$ are stored.

Using the DNA strands, the following eight DNA operations are allowed on the computational model. Since the eight operations are implemented with a constant

number of biological steps for DNA strands [14], we assume that the complexity of each operation is $O(1)$. (See [7] for details of the operations.)

(1) *Merge*: Given two test tubes T_1, T_2, $Merge(T_1, T_2)$ stores the union $T_1 \cup T_2$ in T_1.

(2) *Copy*: Given a test tube T_1, $Copy(T_1, T_2)$ produces a test tube T_2 with the same contents as T_1.

(3) *Detect*: Given a test tube T, $Detect(T)$ outputs "yes" if T contains at least one strand, otherwise, $Detect(T)$ outputs "no".

(4) *Separation*: Given a test tube T_1 and a set of strings X, $Separation(T_1, X, T_2)$ removes all single strands containing one of strings in X from T_1, and produces a test tube T_2 with the removed strands.

(5) *Selection*: Given a test tube T_1 and an integer L, $Selection(T_1, L, T_2)$ removes all single strands, whose length is L, from T_1, and produces a test tube T_2 with the removed strands. (The length of a strand is the number of symbols in the strand.)

(6) *Cleavage*: Given a test tube T and a string of two symbols $\sigma_0\sigma_1$, $Cleavage(T, \sigma_0\sigma_1)$ cuts each double strand containing $\begin{bmatrix} \sigma_0\sigma_1 \\ \overline{\sigma_0\sigma_1} \end{bmatrix}$ in T into two double strands as follows.

$$\begin{bmatrix} \alpha_0\sigma_0\sigma_1\beta_0 \\ \alpha_1\overline{\sigma_0\sigma_1}\beta_1 \end{bmatrix} \Rightarrow \begin{bmatrix} \alpha_0\sigma_0 \\ \alpha_1\overline{\sigma}_0 \end{bmatrix}, \begin{bmatrix} \sigma_1\beta_0 \\ \overline{\sigma}_1\beta_1 \end{bmatrix}$$

(We assume that *Cleavage* cut only double strands at a specific set of sequences.)

(7) *Annealing*: Given a test tube T, $Annealing(T)$ produces all feasible double strands from single strands in T. (The produced double strands are still stored in T after *Annealing*.)

(8) *Denaturation*: Given a test tube T, $Denaturation(T)$ dissociates each double strand in T into two single strands.

In addition to the above, we add the following operation in order to clarify description of this paper. The complexity of the operation is also $O(1)$.

(9) *Empty*: Given a test tube T, $Empty(T)$ sets $T = \phi$.

2.2. *Representation of binary numbers with DNA strands*

In this subsection, we explain a data structure for storing a set of n binary numbers using DNA strands. Let us consider a number x such that $x = \sum_{j=0}^{m-1} x_j * 2^j$, where $x_{m-1}, x_{m-2}, \ldots, x_0$ are binary bits. We assume that the most significant bit x_{m-1} is a sign bit, and a negative number is denoted using two's complement notation. The representation of each bit is the same as that in [7], and is briefly described in the following.

We first define the alphabet Σ as follows.

$$\Sigma = \{A_0, A_1, \ldots, A_n, B_0, B_1, \ldots, B_{m-1}, C_0, C_1, D_0, D_1, 1, 0, \sharp,$$
$$\overline{A}_0, \overline{A}_1, \ldots, \overline{A}_n, \overline{B}_0, \overline{B}_1, \ldots, \overline{B}_{m-1}, \overline{C}_1, \overline{C}_2, \overline{D}_1, \overline{D}_2, \overline{1}, \overline{0}, \overline{\sharp}\}$$

In the above alphabet, $A_0, A_1, \ldots, A_n$ denote addresses of numbers, and $B_0, B_1, \ldots, B_{m-1}$ denote bit positions. In addition, C_0, C_1 and D_0, D_1 are specified symbols cut by *Cleavage*. Symbols "0" and "1" are used to denote values of bits, and "$\sharp$" is a special symbol for *Separation*.

Using the above alphabet, a value of a bit, whose address and bit position are i and j, is represented by a single strand $S_{i,j}$ such that

$$S_{i,j} = D_1 A_i B_j C_0 C_1 V_{i,j} D_0,$$

where $V_{i,j} = 0$ if a value of the bit is 0, otherwise, $V_{i,j} = 1$. We call each $S_{i,j}$ a *memory strand*, and assume that $S_{i,j}(0)$ and $S_{i,j}(1)$ denote memory strands whose values are 0 and 1 as follows.

$$S_{i,j}(0) = D_1 A_i B_j C_0 C_1 0 D_0, \qquad S_{i,j}(1) = D_1 A_i B_j C_0 C_1 1 D_0$$

We use a set of $O(mn)$ different memory strands to denote n binary numbers of m bits, that is, a number x stored in address i is represented by a set of memory strands $\{S_{i,m-1}, S_{i,m-2}, \ldots, S_{i,0}\}$, which denote binary bits $x_{m-1}, x_{m-2}, \ldots, x_0$, respectively. We assume that V_i denotes a value stored in address i as follows.

$$V_i = \sum_{j=0}^{m-1} V_{i,j} * 2^j$$

2.3. *Primitive operations*

In this paper, the three operations, $ValueAssignment$, *Logic* and $Subtraction$, are used as primitive operations.

The $ValueAssignment\_V(T_{input}, T_{output})$ is an operation which assigns the same value $V(\in \{0,1\})$ to all memory strands in the test tube T_{input} and store the result in the test tube T_{output}.

The $Logic(T_{input}, L, T_{output})$ is an operation which executes logic operations. Inputs of the operation is the test tubes T_{input} and L. T_{input} stores memory strands, and L stores some single strands which define pairs of memory strands and kinds of logic operations. Outputs of the logic operations are stored in the test tube T_{output}.

The $Subtraction(T_{input}, R, T_{output})$ is an operation which executes subtractions. Inputs of the operation are the test tubes, T_{input} and R. T_{input} stores memory strands, and R stores some single strands which indicate pairs of memory strands for which subtractions are executed. Outputs of the subtractions are stored in the test tube T_{output}.

For these three primitive operations, the following lemmas are obtained in [7].

Lemma 1. *The* $ValueAssignment\_V(T_{input}, T_{output})$ *can be executed in* $O(1)$ *steps using* $O(1)$ *kinds of different additional DNA strands.* □

Lemma 2. *The* $Logic(T_{input}, L, T_{output})$*, which is for* $O(n)$ *pairs of* m*-bit binary numbers, can be executed in* $O(1)$ *steps using* $O(mn)$ *kinds of different additional DNA strands.* □

Lemma 3. *The* $Subtraction(T_{input}, R, T_{output})$*, which is for* $O(n)$ *pairs of* m*-bit binary numbers, can be executed in* $O(1)$ *steps using* $O(mn)$ *kinds of different additional DNA strands.* □

3. Procedures for computing the maximum

In this section, we propose three procedures for computing the maximum. We assume that an input of the operation is a set of n binary numbers of m bits. The first procedure consists of a repetition of checking on m bit positions. The second procedure consists of a repetition of parallel comparisons of two numbers. The technique is known as a balanced binary tree [4], which is used in parallel algorithms. The third procedure mainly consists of $O(n^2)$ parallel comparisons, and is inspired from a parallel algorithm for computing the maximum in a constant time [18].

3.1. *Input of procedures*

We assume that an input is given by the test tube T_{input} such that

$$T_{input} = \{S_{i,j} \mid 0 \leq i \leq n, 0 \leq j \leq m-1\},$$

where $\{S_{i,j} \mid 0 \leq i \leq n-1, 0 \leq j \leq m-1\}$ is a set of memory strands which denote n input binary numbers and $\{S_{n,j} \mid 0 \leq j \leq m-1\}$ is a set of memory strands in which an output of the procedure is stored. We assume that all input numbers are positive and distinct to simplify the following description.

3.2. *An $O(m)$ step procedure*

3.2.1. *The outline of procedure*

The first procedure consists of a repetition of checking on m bit positions, and works in $O(m)$ steps. For example, we assume that an input of the procedure is $\{00101, 01111, 11011, 10100, 00011, 11001, 01010, 11000\}$. In the first step of the procedure, we check the left-most bit for all binary numbers. Since there are four numbers whose left-most bit is 1, we can determine that the left-most bit of the maximum is 1, and remove every binary number such that a value of its left-most bit is 0. In the second step, we check the next bit for the remaining binary numbers, 11011, 10100, 11001, 11000, and determine that a value of the next bit is also 1. We repeat the above steps for all bit positions, and determine that the maximum is 11011.

We describe an idea of the procedure more precisely. We assume that $\{x_0, x_1, \ldots, x_{n-1}\}$ is a set of n input binary numbers such that $x_i = \sum_{j=0}^{m-1} x_{i,j} * 2^j$, where each $x_{i,j}$ is a binary value, and x_n is a variable in which the maximum of the input is stored. At the beginning, we assume that all input numbers are "winners". In the procedure, we first check the m-th bits of all binary numbers, which are $\{x_{i,m-1} \mid 0 \leq i \leq n-1\}$. If there exists at least one $x_{i,m-1}$ such that $x_{i,m-1} = 1$, then we set $x_{n,m-1} = 1$, otherwise, we set $x_{n,m-1} = 0$. We identify all numbers x_j such that $x_{j,m-1} = x_{n,m-1}$ as winners, and also identify the other numbers as "losers". (The winner is identified by a specified single strand.) We repeat the above check from the $m-1$-th bit to the first bit. After the repetition, there exists an unique winner, which is the maximum of the input, and the number is stored in x_n.

In the procedure, we mainly use the following test tubes.

T_{win}**:** Single strands which denote winners are stored in T_{win}.

T_n**:** Memory strands which denote an output binary value, that is, $\{S_{n,j} \mid 0 \leq j \leq m-1\}$, are stored in T_n.

T_0, T_1**:** Single strands, which denote all numbers whose checked bits are 0 and 1, are stored in T_0 and T_1, respectively.

We now describe an overview of the procedure $MaxOperation1$, which computes the maximum of n binary numbers of m bits in $O(m)$ steps.

Procedure $MaxOperation1$

Step 1: Repeat the following substeps from $j = m-1$ to $j = 0$.

(1-1) Remove memory strands which denote the j-th bit of each number from T_{input}, and store the removed memory strands to T_{win}. Then, remove memory strands which denote j-th bit of an output number from T_{win}, and store the removed memory strands in T_n.

(1-2) Move all memory strands, which are identified as winners, from T_{win} to T_0, and return the other memory strands, which are identified as losers, from T_{win} to T_{input}.

(1-3) Remove all memory strands, whose j-th bit is 1, from T_0, and store the removed strands to T_1.

(1-4) Detect memory strands in T_1. If there exists at least one memory strand in T_1, identify all memory strands in T_1 as winners and assign "1" to all memory strands in T_n. Otherwise, identify all memory strands in T_0 as winners and assign "0" to all memory strands in T_n. Then, return memory strands in T_n to T_{input}.

(1-5) Return all memory strands in T_0 and T_1 to T_{input}.

(End of the procedure)

3.2.2. *Detail of the procedure*

We now describe details of the procedure step by step. In the following description, the following two test tubes are used.

T_{tmp}**:** DNA strands are temporarily stored in T_{tmp}.
T_{trash}**:** Unnecessary strands are discarded into T_{trash}.
$T_{address}$**:** A role of $T_{address}$ is explained in the following.

First of all, we set a test tube T_{win} as follows.

$$T_{win} = \{\overline{\#D_0D_1A_i} \mid 0 \leq i \leq n-1\}$$

The single strand $\overline{\#D_0D_1A_i}$ in T_{win} means that the value V_i is a candidate for the maximum. In other words, $\overline{\#D_0D_1A_i}$ is in T_{win} if and only if a set of memory strands $\{S_{i,j} \mid 0 \leq j \leq m-1\}$ is a winner.

Substep (1-1) consists of the following operations.

Substep (1-1)

$Empty(T_{tmp})$, $Empty(T_n)$

$Separation(T_{input}, \{B_j\}, T_{tmp})$
$Merge(T_{win}, T_{tmp})$
$Separation(T_{tmp}, \{A_n\}, T_n)$

In Substep (1-2), we first merge a set of single strands $\{\#D_0\}$ to T_{win}, and execute *Annealing* and *Denaturation* for T_{win}. Then, using *Separation* with the symbol $\#D_0D_1$ for T_{win}, we select memory strands which are identified as winners. All of the selected memory strands are moved to T_0, and the others are returned into T_{input}. This substep consists of the following operations, and is illustrated in Figure 1 (a).

Substep (1-2)

$Empty(T_{tmp})$, $Empty(T_0)$
$Merge(T_{win}, \{\#D_0\})$
$Annealing(T_{win})$
$Denaturation(T_{win})$
$Separation(T_{win}, \{\#D_0D_1\}, T_0)$
$Separation(T_{win}, \{D_1\}, T_{tmp})$
$Merge(T_{input}, T_{tmp})$

Substep (1-3) consists of the following two operations.

Substep (1-3)

$Empty(T_1)$
$Separation(T_0, \{C_11D_0\}, T_1)$

In Substep (1-4), we first judge whether there exists a DNA strand in a test tube T_1 by *Detect*. If the output is "yes", a value of the j-th bit of the maximum is set to 1, otherwise the value is set to 0. This operation is performed by the

primitive operation, $ValueAssignment$, which is described in Section 2. Then, using an additional test tube $T_{address}$ given below, each single strand $\overline{\#D_0D_1A_i}$ is moved to T_{win} according to the value of j-th bit of the maximum.

$$T_{address} = \{\overline{\#D_0}, \overline{D_1A_i} \mid 0 \leq i \leq n-1\}$$

This substep mainly consists of operations *Annealing*, *Denaturation*, and *Separation*. After Substep (1-4), the single strand $\overline{\#D_0D_1A_i}$ is stored in T_{win} if and only if V_i is still a candidate for the maximum. This substep is illustrated in Figure 1 (b).

Substep (1-4)

$Empty(T_{win})$, $Empty(T_{tmp})$
if ($Detect(T_1)$ is"yes") {
  $ValueAssignment\_1(T_n, T_{tmp})$

  $Merge(T_1, T_{address})$
  $Annealing(T_1)$
  $Denaturation(T_1)$
  $Separation(T_1, \{\overline{\#D_0D_1}\}, T_{win})$
}
else {
  $ValueAssignment\_0(T_n, T_{tmp})$

  $Merge(T_0, T_{address})$
  $Annealing(T_0)$
  $Denaturation(T_0)$
  $Separation(T_0, \{\overline{\#D_0D_1}\}, T_{win})$
}
$Merge(T_{input}, T_{tmp})$

In Substep (1-5), we return all memory strands in T_0 and T_1 to T_{input}. The substep consists of the following operations, and is illustrated in Figure 1 (c).

Substep (1-5)

$Empty(T_{tmp})$
$Merge(T_0, T_1)$
$Merge(T_0, \{\overline{D_0D_1}\})$
$Annealing(T_0)$
$Cleavage(T_0, D_0D_1)$
$Denaturation(T_0)$
$Separation(T_0, \{\#D_0, \overline{D_0}, \overline{D_1}\}, T_{trash})$
$Merge(T_{input}, T_0)$

$$\left[\frac{\#D_0D_1A_iB_jC_0C_1V_{i,j}D_0}{\#D_0D_1A_i}\right] \Rightarrow \#D_0S_{i,j}$$

(a)

$$\left[\frac{\#D_0D_1A_iB_jC_0C_1V_{i,j}D_0}{\#D_0D_1A_i}\right] \Rightarrow \overline{\#D_0D_1A_i}$$

(b)

$$\left[\frac{\#D_0D_1A_iB_jC_0C_1V_{i,j}D_0}{\overline{D_0D_1}}\right]$$

$$\Rightarrow \left[\frac{\#D_0}{\overline{D_0}}\right], \left[\frac{D_1A_iB_jC_0C_1V_{i,j}D_0}{\overline{D_1}}\right]$$

$$\Rightarrow S_{i,j}$$

(c)

Fig. 1. DNA strands in each step: (a) Substep (1-2), (b) Substep (1-4), and (c) Substep (1-5).

We now consider complexity of the above procedure. Each substep consists of a constant number of operations, which are described in Section 2. In addition, $O(m + n)$ kinds of DNA strands are used in the procedure. Then, we obtain the following theorem.

Theorem 4. *Procedure* $MaxOperation1$*, which computes the maximum of* n *numbers of* m *bits, runs in* $O(m)$ *steps using* $O(m+n)$ *different additional DNA strands.* □

3.3. *An* $O(\log n)$ *step procedure*

3.3.1. *The outline of procedure*

The second procedure consists of a repetition of parallel comparisons of two numbers, and works in $O(\log n)$ steps. For example, we assume that an input of the procedure is $\{5, 3, 15, 25, 27, 10, 20, 24\}$. In the first step of the procedure, we compare four pairs of numbers $(5, 3)$, $(15, 25)$, $(27, 10)$, $(20, 24)$. In the first comparisons, the numbers 5, 25, 27, and 24 win the comparisons. Next, we compare two pairs of numbers $(5, 25)$, $(27, 24)$. Then, the numbers 25 and 27 win the comparisons. In the last step, we compare $(27, 25)$, and then, 27 is a winner, and is also the maximum in the input.

The technique is known as a balanced binary tree [4]. To simplify the description, we assume that $\{x_0, x_1, \ldots, x_{n-1}\}$ is a set of input numbers and $n = 2^k$ for a positive

integer k[a]. In the first step of the procedure, we compare each pair of $(x_i, x_{i+\frac{n}{2}})$ for $0 \leq i \leq \frac{n}{2} - 1$, and set $x_i = \max\{x_i, x_{i+\frac{n}{2}}\}$. To compare a pair of two numbers, we use the procedure for subtraction, which is described in Section 2. Next, we compare each pair of $(x_i, x_{i+\frac{n}{4}})$ for $0 \leq i \leq \frac{n}{4} - 1$, and set $x_i = \max\{x_i, x_{i+\frac{n}{4}}\}$. We repeat the above comparison $\log_2 n$ times, and then, the maximum is set to x_0.

In the procedure, we mainly use the following test tubes.

T_{tmp}**:** DNA strands are temporarily stored in T_{tmp}.
T_{sub}**:** Memory strands which denote results of subtractions are stored in T_{sub}.
T_{sign}**:** Single strands which denote sign bits are stored in T_{sign}.

We now describe an overview of the procedure $MaxOperation2$, which computes the maximum of n binary numbers of m bits in $O(\log n)$ steps.

Procedure $MaxOperation2$
Step 1: Copy T_{input} to T_{tmp}.
Step 2: Set $k = n$, and repeat the following substeps until $k = 1$.
(2-1) Compute subtraction of each pair of numbers $(V_i, V_{i+\frac{k}{2}})$ for $0 \leq i \leq \frac{k}{2} - 1$. The results of the subtraction are stored in T_{sub} using additional single strands.
(2-2) Remove single strands which denote m-th bits of the results from T_{sub}, and store the removed strands to T_{sign}. Then, generate single strands which denote losers of comparisons using T_{sign}.
(2-3) Using the generated strands, remove memory strands which denote losers from T_{tmp}, and store memory strands which denote winners in T_{tmp}. (In other words, this substep sets $V_i = \max\{V_i, V_{i+\frac{k}{2}}\}$ for $0 \leq i \leq \frac{k}{2} - 1$.) Then, set $k = \frac{k}{2}$.
Step 3: Copy V_0 to V_n.
(End of the procedure)

3.3.2. *Detail of the procedure*

We now describe details of the procedure. In the following description, the following test tubes are also used.

T'_{tmp}, T''_{tmp}**:** DNA strands are temporarily stored in T'_{tmp} and T''_{tmp}
T_{lose}, T'_{lose}**:** Single strands which denote losers of the comparisons are stored in T_{lose} and T'_{lose}.
T_{trash}**:** Unnecessary strands are discarded into T_{trash}.

Step 1 simply consists of the following operations.
Step 1

[a]In cast of $n \neq 2^k$, we add dummy input numbers, whose values are $-\infty$, so that $n = 2^k$.

$Empty(T_{tmp})$
$Copy(T_{input},\ T_{tmp})$

We next consider Step 2. In Substep (2-1), we compute subtraction $V_i - V_{i+\frac{n}{2}}$ for each i such that $0 \leq i \leq \frac{n}{2} - 1$. The result of the subtraction is stored in a test tube T_{sub}. This subtraction is performed by *SubtractionOperation*, which is described in Section 2, in $O(1)$ steps using $O(mn)$ kinds of DNA strands. All pairs of the subtraction are indicated by a test tube R, which stores some single strands indicating pairs $(V_i, V_{i+\frac{n}{2}})$ for $0 \leq i \leq \frac{n}{2} - 1$. (We assume that the test tube R is prepared in advance of the procedure.)

Substep (2-1)

$SubtractionOperation(T_{tmp}, R, T_{sub})$

After this substep, T_{sub} contains the following memory strands.

$$T_{sub} = \{S_{i,j} \mid 0 \leq i \leq \frac{n}{2} - 1,\ 0 \leq j \leq m-1,\ V_i = V_i - V_{i+\frac{n}{2}}\}$$

In the first step of Substep (2-2), we move memory strands $S_{i,m-1}$ $(0 \leq i \leq \frac{n}{2}-1)$, which denote the sign bits, from T_{sub} to T_{sign} using *Separation*. Then, if $V_i < V_{i+\frac{k}{2}}$, a memory strand $S_{i,m-1}(1) = D_1 A_i B_{m-1} C_0 C_1 1 D_0$ is in T_{sign}, otherwise, a memory strand $S_{i,m-1}(0) = D_1 A_i B_{m-1} C_0 C_1 0 D_0$ is in T_{sign}. To distinguish the above cases, we merge the following test tube T_{lose} with T_{sign}.

$$T_{lose} = \{D_1, \overline{S_{i,m-1}(1) D_1 \# D_0 D_1 A_i} \mid 0 \leq i \leq n-1\}$$

We execute operations, *Annealing*, *Cleavage*, *Denaturation* and *Separation* for the above T_{lose}. We describe details of Substep (2-2) below. (This substep is illustrated in Fig. 2 (a).)

Substep (2-2)

$Separation(T_{sub},\ B_{m-1},\ T_{sign})$

$Merge(T_{lose},\ T_{sign})$
$Annealing(T_{lose})$
$Cleavage(T_{lose},\ D_0 D_1)$
$Denaturation(T_{lose})$
$Separation(T_{lose}, \{D_1, C_0 C_1, \overline{C_0 C_1}\},\ T_{trash})$

After the above substep, T_{lose} contains the single strand $\overline{D_1 \# D_0 D_1 A_i}$ if and only if V_i is a loser of the comparison $(V_i, V_{i+\frac{n}{2}})$, that is, T_{lose} becomes as follows.

$$T_{lose} = \{\overline{D_1 \# D_0 D_1 A_i} \mid V_i < V_{i+\frac{n}{2}},\ 0 \leq i \leq \frac{n}{2} - 1\}$$

We finally describe Substep (2-3) and Step 3. In Substep (2-3), we first copy test tubes T_{tmp} and T_{lose} to T'_{tmp} and T'_{lose}, respectively. Then, we execute the primitive operation *Logic* defined by a truth table in Fig. 3 for T'_{tmp}. (A test tube L is used to define the truth table in *Logic*.) This *Logic* means assignments $V_i = V_{i+\frac{n}{2}}$ for $0 \leq i \leq \frac{n}{2} - 1$. Next, we remove memory strands which denote losers of the

comparisons from T_{tmp}, and move memory strand which denote winners of the comparisons from T'_{tmp} to T_{tmp}. The removal and the movement are realized using the single strand $\overline{D_1 \# D_0 D_1 A_i}$, which are stored in T_{lose} and T'_{lose}, and is illustrated in Fig. 2 (b). Finally, we cut the single strand and remove unnecessary single strands to obtain memory strands using *Cleavage*.

We summarize operations in Substep (2-3) below.

Substep (2-3)

$Empty(T'_{tmp})$, $Copy(T_{tmp}, T'_{tmp})$
$Merge(T_{lose}, \{D_1 \# D_0\})$
$Empty(T'_{lose})$, $Copy(T_{lose}, T'_{lose})$

$Logic(T'_{tmp}, L, T'_{tmp})$

$Merge(T_{tmp}, T_{lose})$
$Annealing(T_{tmp})$
$Denaturation(T_{tmp})$
$Separation(T_{tmp}, \{\#, \overline{\#}\},\ T_{trash})$

$Merge(T'_{tmp}, T'_{lose})$
$Annealing(T'_{tmp})$
$Denaturation(T'_{tmp})$
$Empty(T''_{tmp})$
$Separation(T'_{tmp}, \{D_1 \# D_0\},\ T''_{tmp})$
$Merge(T''_{tmp}, \{\overline{D_0 D_1}\})$
$Annealing(T''_{tmp})$
$Cleavage(T''_{tmp}, D_0 D_1)$
$Denaturation(T''_{tmp})$
$Separation(T''_{tmp}, \{\#, \overline{D_0}, \overline{D_1}\}, T_{trash})$
$Merge(T_{tmp}, T''_{tmp})$

In Step 3, V_0 is copied to V_n using *Logic*. The copy is indicated by a test tube L'. The L' defines a pair (V_0, V_n) of memory strands, and a kind of logic operations such that $V_n = V_0$. (We omit details of strands in L'.)

Step 3

$Empty(T_{input})$
$Logic(T_{tmp},\ L',\ T_{input})$

We now consider complexity of the above procedure. Each substep consists of a constant number of operations, which are described in Section 2. In addition, $O(mn)$ kinds of strands are used in the procedure. Therefore, we obtain the following theorem.

Theorem 5. *Procedure* $MaxOperation2$*, which computes the maximum of* n *num-*

$$\left[\frac{S_{i,m-1}(1)D_1}{S_{i,m-1}(1)D_1\#D_0D_1A_i}\right]$$

$$\Rightarrow \left[\frac{S_{i,m-1}(1)}{S_{i,m-1}(1)}\right], \left[\frac{D_1}{D_1\#D_0D_1A_i}\right]$$

$$\Rightarrow \overline{D_1\#D_0D_1A_i}$$

(a)

$$\left[\frac{D_1\#D_0D_1A_iB_jC_0C_1V_{i,j}D_0}{\overline{D_1\#D_0D_1A_i}}\right] \Rightarrow D_1\#D_0S_{i,j}$$

(b)

Fig. 2. DNA strands in *MaxOperation*2: (a) Substep (2-2), (b) Substep (2-3).

input		output	
$V_{i,j}$	$V_{i+\frac{n}{2},j}$	$V_{i,j}$	$V_{i+\frac{n}{2},j}$
0	0	0	0
0	1	1	1
1	0	0	0
1	1	1	1

Fig. 3. A truth table for *Logic* in Substep (2-3).

bers of m *bits, runs in* $O(\log n)$ *steps using* $O(mn)$ *different additional DNA strands.* □

3.4. *An* $O(1)$ *step procedure*

3.4.1. *The outline of procedure*

The third procedure consists of $O(n^2)$ parallel comparisons of two numbers, and works in $O(1)$ steps. For example, we assume that an input of the procedure is $\{5, 15, 27, 20\}$. In the first step of the procedure, we compare all pairs of two numbers in the set, that is, (5,5), (5,15), (5,27), (5,20), (15,5), (15,15), (15,27), (15,20), (27,5), (27,15), (27,20), (27,27), (20,5), (20,15), (20,20), (20,27). In these comparisons, the number 27 wins all comparisons, and the winner is the maximum of the input.

We describe an idea of the procedure more precisely. Let $\{x_0, x_1, \ldots, x_{n-1}\}$ be a set of input numbers. In the procedure, we concurrently compare n^2 pairs such that (x_p, x_q) for $0 \leq p \leq n-1$, $0 \leq q \leq n-1$. Each comparison is executed by

subtraction, which is described in the second procedure. In the comparisons, the maximum of input numbers is an unique number which wins all comparisons. (We assume that both numbers win if two values are the same.) We execute the Boolean AND operation for the result of the comparisons, and select the number whose result of the AND operation is "win". The AND operation is realized using a long single strand whose length is in proportion to n.

In the procedure, we mainly use the following test tubes.

T_{sub}, T_{sign}: T_{sub} and T_{sign} play the same roles as in $MaxOperation2$.
T_{win}: Single strands which denote winners of the comparisons are stored in T_{win}.
T_{max}: Memory strands which win all comparisons are stored in T_{max}.

Procedure $MaxOperation3$

Step 1: Compute subtraction of each pair of numbers (V_p, V_q) for $0 \leq p \leq n-1$, $0 \leq q \leq n-1$. The results of the subtraction $V_p - V_q$ is stored in memory strands which denote V_{p*n+j} in a test tube T_{sub}.

Step 2: Remove single strands which denote the sign bits of the numbers from T_{sub}, and store the removed strands to T_{sign}. Then, generate single strands $\overline{\alpha_{p,q}}$ if $V_p \geq V_q$ for $0 \leq p \leq n-1$, $0 \leq q \leq n-1$, using T_{sign}, and store the single strands to T_{win}.

Step 3: For each address i $(0 \leq i \leq n-1)$, concatenate single strands $\alpha_{i,j}$ $(0 \leq j \leq m-1)$ in T_{win}. (We obtain the single strand $\alpha_{i,0}\alpha_{i,1}\ldots\alpha_{i,n-1}$, whose length is in proportion to n, if and only if V_i is the maximum of the input.) Then, move the single strand, whose length is in proportion to n, from T_{win} to T_{max}.

Step 4: Using a single strand in T_{max}, separate memory strands, whose value is the maximum, from T_{input}. Then, copy a value of the memory strands to V_n.

(End of the procedure)

3.4.2. *Detail of the procedure*

We now describe details of the procedure. The following three test tubes are also used in the description.

T_{trash}: Unnecessary strands are discarded into T_{trash}.
$T_{connect}$, $T_{address}$: Roles of $T_{connect}$ and $T_{address}$ are explained in the following.

In Step 1, we execute subtractions for n^2 pairs of numbers in parallel, and store the results in the test tube T_{sub} given below.

$$T_{sub} = \{S_{(p,q),j} \mid 0 \leq p \leq n-1, 0 \leq q \leq n-1, 0 \leq j \leq m-1, V_{(p,q)} = V_p - V_q\}$$

The address $A_{(p,q)}$ of the memory strands $S_{(p,q),j}$ $(0 \leq j \leq m-1)$, which stores the result of the subtraction $V_p - V_q$, is given by $(p,q) = p*n+j$. In other words, the result of $V_p - V_q$ is stored in V_i such that $i = p*n+j$. This subtraction is performed

in $O(1)$ steps using $O(mn^2)$ kinds of DNA strands by *Subtraction* described in Section 2. All pairs of the subtraction are indicated by a test tube R, which stores some single strands which indicate pairs (V_p, V_q) for $0 \leq p \leq n-1$, $0 \leq q \leq n-1$. (We assume that the test tube R is prepared in advance of the procedure.)

Step 1

$Subtraction(T_{input}, R, T_{sub})$

Step 2 is similar to Substep (2-2) in $MaxOperation2$. We move memory strands $S_{(p,q),m-1}$ $(0 \leq p \leq n-1, 0 \leq q \leq n-1)$, which denote sign bits, from T_{sub} to T_{sign} using *Separation*. Then, we merge the following test tube T_{win} with T_{sign}.

$$T_{win} = \{D_1, \overline{S_{(p,q),m-1}(0)D_1A_pA_q\#A_pA_qD_0} \mid 0 \leq p \leq n-1, 0 \leq q \leq n-1\}$$

We execute operations, *Annealing*, *Cleavage*, *Denaturation* and *Separation* for T_{sign}. We describe details of Step 2 below. (The step is illustrated in Fig. 4 (a).)

Step 2

$Empty(T_{sign})$
$Separation(T_{sub}, \{B_{m-1}\}, T_{sign})$
$Merge(T_{win}, T_{sign})$
$Annealing(T_{win})$

$Cleavage(T_{win}, D_0D_1)$
$Denaturation(T_{win})$
$Separation(T_{win}, \{D_1, C_0C_1, \overline{C_0C_1}\}, T_{trash})$

After the above step, T_{win} contains a single strand $\overline{D_1A_pA_q\#A_pA_qD_0}$ if and only if V_p is a winner of the comparison (V_p, V_q), that is, T_{win} becomes as follows.

$$T_{win} = \{\overline{D_1A_pA_q\#A_pA_qD_0} \mid 0 \leq p \leq n-1, 0 \leq q \leq n-1, V_p \geq V_q\}$$

In Step 3, we first merge the following test tube $T_{connect}$ with a test tube T_{win}.

$$\begin{aligned} T_{connect} = &\{D_1A_pA_0\#, A_pA_{n-1}D_0, \mid 0 \leq p \leq n-1\} \\ &\cup\{A_pA_qD_0D_1A_pA_{q+1}\# \mid 0 \leq p \leq n-1, 0 \leq q \leq n-2\} \end{aligned}$$

The role of single strands in $T_{connect}$ is as follows. If V_p is the maximum, T_{win} contains single strands $\{\overline{D_1A_pA_q\#A_pA_qD_0} \mid (0 \leq q \leq n-1)\}$. We execute *Annealing* for T_{win}. Then, the following double strand, whose length is in proportion to n, is obtained if and only if V_i is the maximum. (We assume that $\alpha_{p,q} = D_1A_pA_q\#A_pA_qD_0$.)

$$\begin{aligned} &\left[\frac{D_1A_iA_0\#A_iA_0D_0D_1A_iA_1\#A_iA_1D_0 \cdots D_1A_iA_{n-1}\#A_iA_{n-1}D_0}{D_1A_iA_0\#A_iA_0D_0D_1A_iA_1\#A_iA_1D_0 \cdots D_1A_iA_{n-1}\#A_iA_{n-1}D_0}\right] \\ &= \left[\frac{\alpha_{i,0}\alpha_{i,1} \cdots \alpha_{i,n-1}}{\alpha_{i,0}\alpha_{i,1} \cdots \alpha_{i,n-1}}\right] \end{aligned}$$

$$\left[\frac{S_{(p,q),m-1}(0)D_1}{S_{(p,q),m-1}(0)D_1A_pA_q\#A_pA_qD_0}\right]$$

$$\Rightarrow \left[\frac{S_{(p,q),m-1}(0)}{S_{(p,q),m-1}(0)}\right], \left[\frac{D_1}{D_1A_pA_q\#A_pA_qD_0}\right]$$

$$\Rightarrow \overline{D_1A_pA_q\#A_pA_qD_0}$$

(a)

$$\overline{D_1A_iA_0\#A_iA_0D_0}, \overline{D_1A_iA_1\#A_iA_1D_0}, \cdots, \overline{D_1A_iA_{n-1}\#A_iA_{n-1}D_0}$$

$$\Rightarrow \left[\frac{D_1A_iA_0\#A_iA_0D_0D_1A_iA_1\#A_iA_1D_0\cdots D_1A_iA_{n-1}\#A_iA_{n-1}D_0}{D_1A_iA_0\#A_iA_0D_0D_1A_iA_1\#A_iA_1D_0\cdots D_1A_iA_{n-1}\#A_iA_{n-1}D_0}\right]$$

$$\Rightarrow \frac{D_1A_iA_0\#A_iA_0D_0D_1A_iA_1\#A_iA_1D_0\cdots D_1A_iA_{n-1}\#A_iA_{n-1}D_0,}{D_1A_iA_0\#A_iA_0D_0D_1A_iA_1\#A_iA_1D_0\cdots D_1A_iA_{n-1}\#A_iA_{n-1}D_0}$$

(b)

Fig. 4. DNA strands in $MaxOperation3$: (a) Step 2 and (b) Step 3.

We execute an operation *Denaturation*, and then, separate single strands, whose length are kn, from T_{win} to T_{max}, where k is a length of a single strand $\alpha_{p,q}$. Step 3 consists of the following operations. (The step is illustrated in Fig. 4 (b).)

Step 3

$Merge(T_{win},\ T_{connect})$
$Annealing(T_{win})$
$Denaturation(T_{win})$
$Selection(T_{win}, kn, T_{max})$

In Step 4, we first separate two sets of memory strands to a test tube T_{max}. The one is a set of memory strands whose values denote the maximum, and the other is a set of memory strands whose addresses are A_n. (Recall that an output value is stored in the latter set of memory strands.) The separation of the former set of memory strands is executed using the following test tube $T_{address}$. (The step is illustrated in Fig. 5 (a).)

$$T_{address} = \{\overline{\#D_0D_1},\ \overline{A_i} \mid 0 \leq i \leq n-1\}$$

The single strands in $T_{address}$ are used to detect an address in which the maximum is stored. We merge $T_{address}$ with T_{win}, and execute *Annealing* and *Denaturation*. After the operations, the single strand $\#D_0D_1A_i$ is stored in T_{max} if and only if A_i is the address in which the maximum is stored. The following operations for

$$\left[\frac{D_1A_iA_0\#A_iA_0D_0D_1A_iA_1\#A_iA_1D_0\cdots D_1A_iA_{n-1}\#A_iA_{n-1}D_0,}{\#D_0D_1A_i}\right]$$

$$\Rightarrow \overline{\#D_0D_1A_i}$$

(a)

$$\cdots,\left[\frac{\#D_0D_1A_iB_jC_0C_1V_{i,j}D_0}{\#D_0D_1A_i}\right],\cdots,\left[\frac{\#D_0D_1A_nB_jC_0C_1V_{n,j}D_0}{\#D_0D_1A_n}\right]$$

$$\Rightarrow \#D_0D_1A_iB_jC_0C_1V_{i,j}D_0,\quad \#D_0D_1A_nB_jC_0C_1V_{n,j}D_0$$

$$\Rightarrow \left[\frac{\#D_0D_1A_iB_jC_0C_1V_{i,j}D_0}{\#D_0D_1A_i}\right],\quad \left[\frac{\#D_0D_1A_nB_jC_0C_1V_{n,j}D_0}{\#D_0D_1A_n}\right]$$

$$\Rightarrow \left[\frac{\#D_0}{\#D_0}\right],\left[\frac{D_1A_iB_jC_0C_1V_{i,j}D_0}{D_1A_i}\right],\quad \left[\frac{\#D_0}{\#D_0}\right],\left[\frac{D_1A_nB_jC_0C_1V_{n,j}D_0}{D_1A_n}\right]$$

$$\Rightarrow D_1A_iB_jC_0C_1V_{i,j}D_0,\quad D_1A_nB_jC_0C_1V_{n,j}D_0$$

(b)

Fig. 5. DNA strands in $MaxOperation3$ in Step 4.

the separation are executed using a constant number of operations. (The step is illustrated in Fig. 5 (b).)

Next, we execute *Logic* for the separated memory strands. In this case, *Logic* copies the maximum value to V_n. We notice that parallel assignments may occurs in the test tube because the address in which the maximum is stored is not determined before the procedure. However, parallel assignments to V_n do not occur since there exist only the above two sets of memory strands in the test tube. We assume that the assignments are indicated by a test tube L. The L defines a pair (V_i, V_n) of memory strands for $0 \leq i \leq n-1$, and a kind of logic operations such that $V_n = V_i$. (We omit details of strands in L.)

We now summarize Step 4 in the following.

Step 4

$Separation(T_{max}, \{\overline{\#}\}, T_{trash})$
$Merge(T_{max},\ T_{address})$
$Annealing(T_{max})$
$Denaturation(T_{max})$

$Empty(T_{tmp})$
$Separation(T_{max}, \{\overline{\#D_0D_1}\},\ T_{tmp})$
$Merge(T_{input},\ T_{tmp})$
$Merge(T_{input}, \{\#D_0\})$
$Annealing(T_{input})$
$Denaturation(T_{input})$
$Separation(T_{input}, \{\#, \overline{\#}, A_n\},\ T_{max})$

$Annealing(T_{max})$
$Cleavage(T_{max},\ D_0D_1)$
$Denaturation(T_{max})$
$Separation(T_{max}, \{\#, \overline{\#}, \overline{D_1}\},\ T_{trash})$

$Empty(T_{tmp})$
$Logic(T_{max}, L, T_{tmp})$
$Merge(T_{input}, T_{tmp})$

We finally consider complexity of the above procedure. All steps consist of a constant number of the operations. In addition, $O(mn^2)$ kinds of DNA strands are used in the procedure. Then, we obtain the following theorem.

Theorem 6. *Procedure MaxOperation3, which computes the maximum of n numbers of m bits, runs in $O(1)$ steps using $O(mn^2)$ different additional DNA strands.* □

4. Conclusions

In this paper, we proposed three procedures for computing the maximum. The first procedure consists of a repetition of checking on m bit positions, and runs in $O(m)$ steps using $O(m + n)$ kinds of DNA strands. The second procedure consists of a repetition of comparisons of two numbers, and runs in $O(\log n)$ steps using $O(mn)$ kinds of DNA strands. The third procedure computes consists of $O(n^2)$ parallel comparisons, and runs in $O(1)$ steps using $O(mn^2)$ kinds of DNA strands.

Although our results are based on a theoretical model, the proposed procedures can be implemented practically since every DNA operation used in the model has been already realized in lab level. Therefore, we believe that our results will play an important role in the future DNA computing.

Acknowledgements

This research was partially supported by the Ministry of Education, Culture, Sports, Science and Technology, Grant-in-Aid for Young Scientists (B), 17700021, 2006.

References

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

[2] L.M. Adleman. Computing with DNA. *Scientific American*, Vol. 279, No. 2, pp. 54–61, 1998.

[3] E.B. Baum and D. Boneh. Running dynamic programming algorithms on a DNA computer. *Proceedings of the Second Annual Meeting on DNA Based Computers*, 1996.

[4] E. Dekel and S. Sahni. Parallel scheduling algorithms. *Operations Research*, 31(1):24–49, 1983.

[5] P. Frisco. Parallel arithmetic with splicing. *Romanian Journal of Information Science and Technology(ROMJIST)*, Vol. 2, No. 3, pp. 113–128, 2000.

[6] A.G. Frutos, Q. Liu, A.J. Thiel, A.M.W. Sanner, A.E. Condon, L.M. Smith, and R.M. Corn. Demonstration of a word design strategy for DNA computing on surfaces. *Nucleic Acids Research*, Vol. 25(23), pp. 4748–4757, 1997.

[7] A. Fujiwara, K. Matsumoto, and W. Chen. Procedures for Logic and Arithmetic Operations with DNA Molecules. *International Journal of Foundations of Computer Science*, Vol. 15, No. 3, pp. 461–476, 2004.

[8] F. Guarnieri, M. Fliss, and C. Bancroft. Making DNA add. *Science*, Vol. 273(5272), pp. 220–223, 1996.

[9] V. Gupta, S. Parthasarathy, and M.J. Zaki. Arithmetic and logic operations with DNA. *Proceedings of the 3rd DIMACS Workshop on DNA Based Computers*, pp. 212–220, 1997.

[10] H. Hug and R. Schuler. DNA-based parallel computation of simple arithmetic. *Proceedings of the 7th International Meeting on DNA Based Computers(DNA7)*, pp. 159–166, 2001.

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

[12] R.B. Merrifield. Solid phase peptide synthesis. I. the synthesis of a tetrapeptide. *Journal of the American Chemical Society*, Vol. 85, pp. 2149–2154, 1963.

[13] Q. Ouyang, P.D. Kaplan, S.Liu, and A. Libchaber. DNA solution of the maximal clique problem. *Science*, Vol. 278, pp. 446–449, 1997.

[14] G. Păun, G. Rozeberg, and A. Salomaa. *DNA computing*. Springer-Verlag, 1998.

[15] Z.F. Qiu and M. Lu. Arithmetic and logic operations for DNA computers. *Proceedings of the Second IASTED International conference on Parallel and Distributed Computing and Networks*, pp. 481–486, 1998.

[16] Z.F. Qiu and M. Lu. Take advantage of the computing power of DNA computers. In *Proceedings of the Third Workshop on Bio-Inspired Solutions to Parallel Processing Problems, IPDPS 2000 Workshops*, pp. 570–577, 2000.

[17] J.H. Reif. Parallel biomolecular computation: Models and simulations. *Algorithmica*, Vol. 25, No. 2-3, pp. 142–175, 1995.

[18] Y. Shiloach and U. Vishkin. Finding the maximum, merging and sorting in a parallel computation model. *Journal of Algorithms*, 2(1):88–102, 1981.

[19] A. Suyama, N. Nishida, K. Kurata, and K. Omagari. Gene expression analysis by DNA computing. *Computational Molecular Biology*, pp. 12–13, 2000.

[20] H. Yoshida and A. Suyama. Solution to 3-SAT by breadth first search. *American Mathematical Society*, pp. 9–22, 2000.