Available online at www.sciencedirect.com
SCIENCE DIRECT
BioSystems 84 (2006) 207–216
BioSystems
www.elsevier.com/locate/biosystems

Procedures for a dynamical system on $\{0, 1\}^n$ with DNA molecules

Dongmei Xiao[a,c], Wenxia Li[b,c,*], Jiang Yu[a,c], Xiaodong Zhang[a,c], Zhizhou Zhang[c], Lin He[c]

[a] *Department of Mathematics, Shanghai Jiao Tong University, Shanghai 200030, PR China*
[b] *Department of Mathematics, East China Normal University, Shanghai 200062, PR China*
[c] *Bio-X DNA Computer Consortium, Shanghai Jiao Tong University, Shanghai 200030, PR China*

Received 14 April 2005; received in revised form 9 November 2005; accepted 23 November 2005

Abstract

In this paper, an improved form of DNA representations of elements in $\{0, 1\}^n$, which was first proposed by Fujiwara et al. [Fujiwara, A., Matsumoto, K., Chen, W., 2004. Procedures for logic and arithmetic operations with DNA molecules. Int. J. Found. Comput. Sci. 15, 461–474], is given. Using this improved representations, a procedure for cycling shift is proposed, and this procedure can be implemented in O(1) lab steps theoretically. Based on the operation for cycling shift, dynamic behavior of an operator on $\{0, 1\}^n$ is investigated by DNA molecules.

Keywords: DNA representation; DNA computing; Symbolic space; Dynamic behavior

1. Introduction

A deoxyribonucleic acid (DNA) is a polymer, which can be strung together from monomers called deoxyribonucleotides (Păun et al., 1998). Distinct nucleotides are detected only with their bases. Those bases are, respectively, abbreviated as A (adenine), G (guanine), C (cytosine) and T (thymine). Under appropriate conditions two strands of DNA can form a double strand, if the respective bases are the Watson–Crick complements of each other—A matches T and C matches G; also the 3′-end matches the 5′-end, e.g., the single strands 5′-ACCTGGATGTAA-3′ and 3′-TGGACCTACATT-5′ can form a double strand. We also call the strand 3′-TGGACCTACATT-5′ as the complementary strand of 5′-ACCTGGATGTAA-3′ and simply denote it by $\overline{\text{ACCTGGATGTAA}}$.

DNA has two important features, which are Watson–Crick complementarity and massive parallelism. As the first work for DNA computing, Adleman (1994) took advantage of these features to present an idea of solving the Hamiltonian path problem (an NP problem) of size n in O(n) steps using DNA molecules. Lipton (1995) demonstrated that Adleman's experiment could be used to determine the NP-complete satisfiability (SAT) problem (the first NP-complete problem). Ouyang et al. (1997) presented a molecule biology-based experimental solution to the maximal clique NP-complete problem. Each of these works are based on performing some basic operations on DNA strands.

* Corresponding author. Tel.: +86 2162233060; fax: +86 2152682621.
E-mail address: wxli@math.ecnu.edu.cn (W. Li)

0303-2647/$ – see front matter © 2005 Elsevier Ireland Ltd. All rights reserved.
doi:10.1016/j.biosystems.2005.11.004

A (test) tube is a set of molecules of DNA, i.e., a multi-set of finite strings over the alphabet {A, C, G, T}. Given a tube, one can perform the following basic 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 a string in X from T_1, and produce a test tube T_2 with the removed strands.
[5] *Discard*: Given a tube T, Discard(T) will discard the tube T.
[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:

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

[7] *Annealing*: Given a test tube T, Annealing(T) produces all feasible double 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.
[9] *Selection*: Given a test tube T_1 and an integer L, Selection(T_1, L, T_2) removes all strands, whose length is L, from T_1, and produces a test tube T_2 with the removed strands.
[10] *Append*: Given a tube T and a short DNA single strand Z, Append(T, Z) will append Z onto the end of every strand in the tube T.
[11] *Read*: Given a tube T, the operation Read(T) is used to describe a single molecule, which is contained in the tube T. Even if T contains many different molecules each encoding a different set of bases, the operation can give an explicit description of exactly one of them.

Since these eleven manipulations are implemented with a constant number of biological steps for DNA strands (Păun et al., 1998), we assume that the complexity of each manipulation is O(1) steps.

In order to apply DNA computing on a wide range of problems, procedures for primitive operations, such as logic or arithmetic operations, are needed. There are some works for primitive operations in DNA computing (Frisco, 2002; Guarnieri et al., 1996; Gupta et al., 1997; Hug and Schuler, 2001). As we know, the key to solve a mathematical problem using DNA molecules is to design appropriate DNA representations according to the problem itself. A canonical DNA representation for a binary number was introduced by Fujiwara et al. (2004). It deals with DNA representations of m binary numbers of n bits. A value of the jth bit of the ith binary number is represented by a single strand $S_{i,j}$ such that

$$S_{i,j} = E_1 N_i B_j C_0 C_1 V_{i,j} E_0, \tag{1}$$

where C_0, C_1, E_0, E_1, B_j, $V_{i,j}$ and $N_i (1 \le i \le m, 1 \le j \le n)$ all are single strands. $V_{i,j}$ takes the single strands 0 or 1, representing the real numbers zero or one, respectively. Single strands C_0, C_1 and E_0, E_1 are special symbols cut by Cleavage. For example, the following sets of single strands:

$$\{E_1N_1B_1C_0C_11E_0,\ E_1N_1B_2C_0C_11E_0,\ E_1N_1B_3C_0C_10E_0\},$$

and

$$\{E_1N_2B_1C_0C_11E_0,\ E_1N_2B_2C_0C_10E_0,\ E_1N_2B_3C_0C_10E_0\}$$

denote 2 ($m = 2$) binary numbers of 3 ($n = 3$) bits: 110 and 100, respectively. Based on the representation by (1), Fujiwara et al. (2004) design a very important operation, denoted by ValueAssinment$_V$(T_{input}, T_{output}), which assigns

the same value $V \in \{0, 1\}$ to each bit in the input tube T_{input} and can be implemented in O(1) biological steps. More exactly, if

$$T_{\text{input}} = \{E_1 N_i B_j C_0 C_1 V_{i,j} E_0 : 1 \le i \le m, 1 \le j \le n\},$$

where $V_{i,j} \in \{0, 1\}$, then ValueAssignment$_V(T_{\text{input}}, T_{\text{output}})$ gives that

$$T_{\text{output}} = \{E_1 N_i B_j C_0 C_1 V E_0 : 1 \le i \le m, 1 \le j \le n\}.$$

By means of the operation of ValueAssignment, Fujiwara et al. (2004) proposed procedures for logic and arithmetic operations with DNA molecules. Since each element of $\{0, 1\}^n$ can be considered a binary number of n bits, the above representation and the operation of ValueAssignment can be used for some matters on $\{0, 1\}^n$, e.g., permutation of the first two terms of an element, i.e., change $(V_1, V_2, V_3, \ldots, V_n)$ into $(V_2, V_1, V_3, \ldots, V_n)$ for $(V_1, V_2, V_3, \ldots, V_n) \in \{0, 1\}^n$. However, a typical operation, the cycling shift which changes $(V_1, V_2, V_3, \ldots, V_n)$ into $(V_2, V_3, V_4, \ldots, V_n, V_1)$ for $(V_1, V_2, V_3, \ldots, V_n) \in \{0, 1\}^n$, needs to implement the operation of ValueAssignment $n - 1$ times. To remedy this shortage, we improve the representation in (1) by setting

$$S_{i,j} = E_1 N_i A_j D_0 D_1 B_j C_0 C_1 V_{i,j} E_0, \tag{2}$$

i.e., using both single strands A_j and B_j to locate a bit on position j. With this improved DNA representation, all implementations described in Fujiwara et al. (2004) keep effective and an operation, denoted by CycleShift, can be designed such that it can carry out the cycling shift in O(1) biological steps. Furthermore, the operation of CycleShift can be used to consider the dynamic behavior of some operators on $\{0, 1\}^n$ with DNA molecules.

Fix a positive integer $n \ge 2$ and let $F : \{0, 1\}^n \to \{0, 1\}^n$ by

$$F(k_1, k_2, k_3, \ldots, k_{n-1}, k_n) = (k^*, k_3, k_4, \ldots, k_n, k_1), \tag{3}$$

where $k^* = k_1 + k_2 \pmod 2$. Let $\mathbb{N}$ be the set of positive integers. For any $\alpha \in \{0, 1\}^n$ the iteration of F is defined in the usual way, i.e., $F^0\alpha = \alpha$ and $F^k\alpha = F(F^{k-1}\alpha)$ for $k \in \mathbb{N}$. To investigate the orbit $F^k\alpha$, $k = 0, 1, 2, \ldots$ of $\alpha \in \{0, 1\}^n$, it concerns where $F^k\alpha$ can arrive. In other words, for any $\beta \in \{0, 1\}^n$ whether or not there exists a $k \in \mathbb{N}$ such that $F^k\alpha = \beta$.

Denote $\gamma(k, \ell) = k + \ell \pmod 2$ for $k, \ell \in \{0, 1\}$. For $(k_1, k_2, \ldots, k_{n-1}, k_n) \in \{0, 1\}^n$, let

$$F_1(k_1, k_2, k_3, \ldots, k_{n-1}, k_n) = (k_2, k_3, k_4, \ldots, k_n, k_1),$$

and

$$F_2(k_1, k_2, k_3, \ldots, k_{n-1}, k_n) = (k_1, \gamma(k_1, k_2), k_3, k_4, \ldots, k_n).$$

Then $F = F_1 \circ F_2$. So for a $(k_1, k_2, \ldots, k_{n-1}, k_n) \in \{0, 1\}^n$, $F(k_1, k_2, \ldots, k_{n-1}, k_n)$ can be evaluated by the operations of ValueAssignment and CycleShift.

This paper is organized as follows. In Section 2, we describe the improved representations of elements in $\{0, 1\}^n$ and propose a procedure for the operation of CycleShift. In Section 3, a procedure is proposed for investigating the dynamic behavior of F. Conclusions are summarized in Section 4.

2. Bit representation and cycling transformation

Let Σ be a set of single strands such that

$$\Sigma = \{A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_n, C_0, C_1, D_0, D_1, E_0, E_1, 1, 0, \#, \bar{A}_1, \bar{A}_2, \ldots, \bar{A}_n, \bar{B}_1, \bar{B}_2, \ldots, \bar{B}_n, \bar{C}_0, \bar{C}_1, \bar{D}_0, \bar{D}_1, \bar{E}_0, \bar{E}_1, \bar{1}, \bar{0}, \bar{\#}\}.$$

Both A_i and B_i, $i = 1, 2, \ldots, n$ are used to denote the ith bit position for an element (or a $0 - 1$ sequence) of $\{0, 1\}^n$. C_0, C_1, D_0, D_1 and E_0, E_1 are the specified symbols cut by Cleavage, that is, Cleavage(T, C_0C_1), Cleavage(T, D_0D_1)

and Cleavage(T, E_0E_1) cut all double strands containing

$$\left[\frac{C_0C_1}{\overline{C_0C_1}}\right], \qquad \left[\frac{D_0D_1}{\overline{D_0D_1}}\right] \quad \text{and} \quad \left[\frac{E_0E_1}{\overline{E_0E_1}}\right]$$

in a test tube T, respectively. Symbols "0" and "1" are used to denote values of bits, and # is a special symbol for Separation.

Let $(V_1, V_2, \ldots, V_n) \in \{0, 1\}^n$. Using the above notations, a value of bit at position j is represented by a single strand S_j, called as a memory strand, such that

$$S_j = E_1A_jD_0D_1B_jC_0C_1V_jE_0, \tag{4}$$

where $V_j = 0$ if a value of the bit is 0, otherwise, $V_j = 1$. For instance, the representation of $(1, 0, 0) \in \{0, 1\}^3$ is

$$\{E_1A_1D_0D_1B_1C_0C_1 1E_0,\ E_1A_2D_0D_1B_2C_0C_1 0E_0,\ E_1A_3D_0D_1B_3C_0C_1 0E_0\}.$$

Compared with that in (2), for simplicity we omit N_i and only use one index j in (4) since we do not consider multiple elements of $\{0, 1\}^n$ simultaneously.

In the following we show a procedure for the operation CycleShift. It is implemented in O(1) lab steps. Let $(V_1, V_2, \ldots, V_n) \in \{0, 1\}^n$. The CycleShift transfers $(V_1, V_2, \ldots, V_n)$ into $(V_2, \ldots, V_n, V_1)$. More exactly, if

$$T_{\text{input}} = \{E_1A_iD_0D_1B_iC_0C_1V_iE_0 | 1 \le i \le n\},$$

where $V_i \in \{0, 1\}$, then after performing CycleShift(T_{input}, T_{output}) we have

$$T_{\text{output}} = \{E_1A_nD_0D_1B_nC_0C_1V_1E_0,\ E_1A_iD_0D_1B_iC_0C_1V_{i+1}E_0 | 1 \le i \le n-1\}.$$

Some auxiliary test tubes are given by

$$T_{\bar{D}} = \{\overline{D_0D_1}\}, \qquad T_{\bar{C}} = \{\overline{C_0C_1}\}, \qquad T_{\bar{A}} = \{E_1A_iD_0; \overline{E_1A_iD_0D_1B_{i+1}C_0}, i = 1, \ldots, n-1\},$$

and

$$T_{\bar{B}} = \{D_1B_iC_0, \overline{E_1A_iD_0D_1B_iC_0}, i = 1, \ldots, n-1\}, \qquad T_{\text{zero}} = \{\overline{C_0C_1}, C_1 0E_0\}, \qquad T_{\text{one}} = \{\overline{C_0C_1}, C_1 1E_0\}.$$

The following CycleShift(T_{input}, T_{output}) is implemented in five steps. For reader's convenience, we show the contents in the tubes after some operations.

Procedure CycleShift(T_{input}, T_{output})
Step 1: Shift subscripts of A's
Separation(T_{input}, $\{A_1\}$, T_{first})
$\Longrightarrow T_{\text{first}} = \{E_1A_1D_0D_1B_1C_0C_1V_1E_0\}$.
Separation(T_{input}, $\{A_n\}$, T_{last})
$\Longrightarrow T_{\text{last}} = \{E_1A_nD_0D_1B_nC_0C_1V_nE_0\}$.
Copy(T_{last}, T_{tmp})
Merge(T_{input}, T_{tmp})
$\Longrightarrow T_{\text{input}} = \{E_1A_iD_0D_1B_iC_0C_1V_iE_0, 2 \le i \le n\}$.
Copy($T_{\bar{D}}$, $T_{\bar{D}^*}$)
Merge(T_{input}, $T_{\bar{D}^*}$)
$\Longrightarrow T_{\text{input}} = \{\overline{D_0D_1}, E_1A_iD_0D_1B_iC_0C_1V_iE_0, 2 \le i \le n\}$.
Annealing(T_{input})

$$\Longrightarrow T_{\text{input}} = \left\{ \left[\begin{array}{c} E_1A_iD_0D_1B_iC_0C_1V_iE_0 \\ \overline{D_0D_1} \end{array}\right], 2 \le i \le n \right\}.$$

Cleavage(T_{input}, D_0D_1)

$$\Longrightarrow T_{\text{input}} = \left\{ \left[\begin{array}{c} E_1A_iD_0 \\ \overline{D_0} \end{array}\right], \left[\begin{array}{c} D_1B_iC_0C_1V_iE_0 \\ \overline{D_1} \end{array}\right], 2 \le i \le n \right\}.$$

Denaturation(T_{input})

$\Longrightarrow T_{\text{input}} = \{\bar{D}_0, \bar{D}_1, E_1A_iD_0, D_1B_iC_0C_1V_iE_0, 2 \le i \le n\}$.

Separation($T_{\text{input}}, \{C_0C_1\}, T_{\text{output}}$)

$\Longrightarrow T_{\text{input}} = \{\bar{D}_0, \bar{D}_1, E_1A_iD_0, 2 \le i \le n\}, \qquad T_{\text{output}} = \{D_1B_iC_0C_1V_iE_0, 2 \le i \le n\}$.

Copy($T_{\bar{A}}, T_{\bar{A}^*}$)

Merge($T_{\text{output}}, T_{\bar{A}^*}$)

$\Longrightarrow T_{\text{output}} = \{E_1A_{i-1}D_0, \overline{E_1A_{i-1}D_0D_1B_iC_0}, D_1B_iC_0C_1V_iE_0, 2 \le i \le n\}$.

Annealing(T_{output})

$$\Longrightarrow T_{\text{output}} = \left\{ \left[\begin{matrix} E_1A_{i-1}D_0D_1B_iC_0C_1V_iE_0 \\ \overline{E_1A_{i-1}D_0D_1B_iC_0} \end{matrix} \right], 2 \le i \le n \right\}.$$

Denaturation(T_{output})

$\Longrightarrow T_{\text{output}} = \{E_1A_{i-1}D_0D_1B_iC_0C_1V_iE_0, \overline{E_1A_{i-1}D_0D_1B_iC_0}, 2 \le i \le n\}$.

Separation($T_{\text{output}}, \{\overline{D_0D_1}\}, T_{\text{tmp}}$)

$\Longrightarrow T_{\text{output}} = \{E_1A_{i-1}D_0D_1B_iC_0C_1V_iE_0, 2 \le i \le n\}, \qquad T_{\text{tmp}} = \{\overline{E_1A_{i-1}D_0D_1B_iC_0}, 2 \le i \le n\}$.

Discard(T_{tmp})

Separation($T_{\text{output}}, \{0\}, T_0$)

$\Longrightarrow T_0 = \{E_1A_{i-1}D_0D_1B_iC_0C_1V_iE_0 | V_i = 0, 2 \le i \le n\}$.

Separation($T_{\text{output}}, \{1\}, T_1$)

$\Longrightarrow T_1 = \{E_1A_{i-1}D_0D_1B_iC_0C_1V_iE_0 | V_i = 1, 2 \le i \le n\}$.

Step 2: Shift subscripts of B's for single strands in tube T_0

Copy($T_{\bar{C}}, T_{\bar{C}^*}$)

Merge($T_0, T_{\bar{C}^*}$)

$\Longrightarrow T_0 = \{\overline{C_0C_1}, E_1A_{i-1}D_0D_1B_iC_0C_1V_iE_0 | V_i = 0, 2 \le i \le n\}$.

Annealing(T_0)

$$\Longrightarrow T_0 = \left\{ \left[\begin{matrix} E_1A_{i-1}D_0D_1B_iC_0C_1V_iE_0 \\ \overline{C_0C_1} \end{matrix} \right] \middle| V_i = 0, 2 \le i \le n \right\}.$$

Cleavage(T_0, C_0C_1)

$$\Longrightarrow T_0 = \left\{ \left[\begin{matrix} E_1A_{i-1}D_0D_1B_iC_0 \\ \overline{C_0} \end{matrix} \right], \left[\begin{matrix} C_1V_iE_0 \\ \overline{C_1} \end{matrix} \right] \middle| V_i = 0, 2 \le i \le n \right\}.$$

Denaturation(T_0)

$\Longrightarrow T_0 = \{E_1A_{i-1}D_0D_1B_iC_0, \bar{C}_0, C_1V_iE_0, \bar{C}_1 | V_i = 0, 2 \le i \le n\}$.

Separation($T_0, \{C_1, \bar{C}_0, \bar{C}_1\}, T_{\text{tmp}}$)

$\Longrightarrow T_0 = \{E_1A_{i-1}D_0D_1B_iC_0 | V_i = 0, 2 \le i \le n\}$.

Copy($T_{\bar{D}}, T_{\bar{D}^*}$)

Merge($T_0, T_{\bar{D}^*}$)

$\Longrightarrow T_0 = \{\overline{D_0D_1}, E_1A_{i-1}D_0D_1B_iC_0 | V_i = 0, 2 \le i \le n\}$.

Annealing(T_0)

$$\Longrightarrow T_0 = \left\{ \left[\begin{matrix} E_1A_{i-1}D_0D_1B_iC_0 \\ \overline{D_0D_1} \end{matrix} \right] \middle| V_i = 0, 2 \le i \le n \right\}.$$

Cleavage(T_0, D_0D_1)

$$\Longrightarrow T_0 = \left\{ \left[\begin{matrix} E_1A_{i-1}D_0 \\ \overline{D_0} \end{matrix} \right], \left[\begin{matrix} D_1B_iC_0 \\ \overline{D_1} \end{matrix} \right] \middle| V_i = 0, 2 \le i \le n \right\}.$$

Denaturation(T_0)

$\Longrightarrow T_0 = \{\bar{D}_0, \bar{D}_1, E_1A_{i-1}D_0, D_1B_iC_0 | V_i = 0, 2 \le i \le n\}$.

Separation(T_0, $\{\bar{D}_0, \bar{D}_1, C_0\}$, T_{tmp})

$\Longrightarrow T_0 = \{E_1 A_{i-1} D_0 | V_i = 0, 2 \le i \le n\}$.

Discard(T_{tmp})

Copy($T_{\bar{B}}$, $T_{\bar{B}^*}$)

Merge(T_0, $T_{\bar{B}^*}$)

$\Longrightarrow T_0 = \{E_1 A_{i-1} D_0 | V_i = 0, 2 \le i \le n\} \cup \{D_1 B_{i-1} C_0, \overline{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0} | 2 \le i \le n\}$.

Annealing(T_0)

$$\Longrightarrow T_0 = \left\{ \left[\begin{array}{c} E_1 A_{i-1} D_0 D_1 B_{i-1} C_0 \\ \hline \overline{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0} \end{array} \right] \middle| \, V_i = 0, 2 \le i \le n \right\},$$

$$\bigcup \left\{ \left[\begin{array}{c} D_1 B_{i-1} C_0 \\ \hline \overline{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0} \end{array} \right] \middle| \, \text{for some other } 2 \le i \le n \right\}.$$

Denaturation(T_0)

$\Longrightarrow T_0 = \{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0, \overline{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0} | V_i = 0, 2 \le i \le n\}$

$\cup \{D_1 B_{i-1} C_0, \overline{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0} | \text{ for some other } 2 \le i \le n\}$.

Separation(T_0, $\{D_0 D_1\}$, T_{tmp})

$\Longrightarrow T_{\text{tmp}} = \{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0, | V_i = 0, 2 \le i \le n\}$.

Discard(T_0)

Merge(T_0, T_{tmp})

$\Longrightarrow T_0 = \{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0, | V_i = 0, 2 \le i \le n\}$.

Copy(T_{zero}, $T_{\text{zero}*}$)

Merge(T_0, $T_{\text{zero}*}$)

$\Longrightarrow T_0 = \{\overline{C_0 C_1}, C_1 0 E_0, E_1 A_{i-1} D_0 D_1 B_{i-1} C_0, | V_i = 0, 2 \le i \le n\}$.

Annealing(T_0)

$$\Longrightarrow T_0 = \left\{ \left[\begin{array}{c} E_1 A_{i-1} D_0 D_1 B_{i-1} C_0 C_1 0 E_0 \\ \hline \overline{C_0 C_1} \end{array} \right] \middle| \, V_i = 0, 2 \le i \le n \right\}.$$

Denaturation(T_0)

$\Longrightarrow T_0 = \{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0 C_1 0 E_0, \overline{C_0 C_1} | V_i = 0, 2 \le i \le n\}$.

Separation(T_0, $\{\overline{C_0 C_1}\}$, $T_{\bar{C}}$)

$\Longrightarrow T_0 = \{E_1 A_{i-1} D_0 D_1 B_{i-1} C_0 C_1 0 E_0, | V_i = 0, 2 \le i \le n\}$.

Step 3: Shift subscripts of B's for single strands in tube T_1 (by the same manipulations as those in Step 2)

Copy($T_{\bar{C}}$, $T_{\bar{C}^*}$); Merge(T_1, $T_{\bar{C}^*}$); Annealing(T_1); Cleavage(T_1, $C_0 C_1$); Denaturation(T_1); Separation(T_1, $\{C_1, \bar{C}_0, \bar{C}_1\}$, T_{tmp}); Copy($T_{\bar{D}}$, $T_{\bar{D}^*}$); Merge(T_1, $T_{\bar{D}^*}$); Annealing(T_1); Cleavage(T_1, $D_0 D_1$); Denaturation(T_1); Separation(T_1, $\{\bar{D}_0, \bar{D}_1, C_0\}$, T_{tmp}); Discard(T_{tmp}); Copy($T_{\bar{B}}$, $T_{\bar{B}^*}$); Merge(T_1, $T_{\bar{B}^*}$); Annealing(T_1); Denaturation(T_1); Separation(T_1, $\{D_0 D_1\}$, T_{tmp}); Discard(T_1); Merge(T_1, T_{tmp}); Copy(T_{one}, $T_{\text{one}*}$); Merge(T_1, $T_{\text{one}*}$); Annealing(T_1); Denaturation(T_1); Separation(T_1, $\{\overline{C_0 C_1}\}$, $T_{\bar{C}}$).

Step 4: Assign the nth bit with value V_1

Separation(T_{first}, $\{1\}$, T_{tmp})

If Detect(T_{first}) is "yes", then

ValueAssignment$_0$(T_{last}, $T_{\text{last}*}$), else

ValueAssignment$_1$(T_{last}, $T_{\text{last}*}$)

$\Longrightarrow T_{\text{last}} = \{E_1 A_n D_0 D_1 B_n C_0 C_1 V_1 E_0\}$.

Step 5: Produce the output T_{output}

Merge(T_1, T_0)

Merge(T_1, T_{last})

Merge(T_{output}, T_1)

$\Longrightarrow T_{\text{output}} = \{E_1A_nD_0D_1B_nC_0C_1V_1E_0,\ E_1A_iD_0D_1B_iC_0C_1V_{i+1}E_0 | 1 \le i \le n-1\}$.

Note that the operation ValueAssignment introduced by Fujiwara et al. (2004) is used in Step 4. Although there is a bit difference for the DNA representations of binary numbers given by (1) and (4), the operation ValueAssignment is still available for our setting. For readers' convenience, we give a description of the operation ValueAssignment as follows. Let

$$T_{\text{input}} = \{E_1A_iD_0D_1B_iC_0C_1V_iE_0 | 1 \le i \le n\},$$

where $V_i \in \{0, 1\}$. Then the operation ValueAssignment$_V$(T_{input}, T_{output}) produces

$$T_{\text{output}} = \{E_1A_iD_0D_1B_iC_0C_1VE_0 | \le i \le n\},$$

where $V \in \{0, 1\}$ and all memory strands are set to the same value V. Let $T_{\bar{C}}$ and T_V be two auxiliary test tubes such that $T_{\bar{C}} = \{\overline{C_0C_1}\}$, $T_V = \{C_1VE_0, \overline{C_0C_1}\}$.

Procedure ValueAssignment$_V$(T_{input}, T_{output})

Step 1: Delete values from memory strands.

Copy($T_{\bar{C}}$, $T_{\bar{C}^*}$)

Merge(T_{input}, $T_{\bar{C}^*}$)

$\Longrightarrow T_{\text{input}} = \{\overline{C_0C_1},\ E_1A_iD_0D_1B_iC_0C_1V_iE_0 | 1 \le i \le n\}$.

Annealing(T_{input})

$$\Longrightarrow T_{\text{input}} = \left\{ \begin{bmatrix} E_1A_iD_0D_1B_iC_0C_1V_iE_0 \\ \overline{C_0C_1} \end{bmatrix} \middle| 1 \le i \le n \right\}.$$

Cleavage(T_{input}, C_0C_1)

$$\Longrightarrow T_{\text{input}} = \left\{ \begin{bmatrix} E_1A_iD_0D_1B_iC_0 \\ \overline{C_0} \end{bmatrix}, \begin{bmatrix} C_1V_iE_0 \\ \overline{C_1} \end{bmatrix} \middle| 1 \le i \le n \right\}.$$

Denaturation(T_{input})

$\Longrightarrow T_{\text{input}} = \{\bar{C}_0, \bar{C}_1, E_1A_iD_0D_1B_iC_0, C_1V_iE_0 | 1 \le i \le n\}$.

Separation(T_{input}, $\{C_1, \bar{C}_0, \bar{C}_1\}$, T_{tmp})

$\Longrightarrow T_{\text{input}} = \{E_1A_iD_0D_1B_iC_0 | 1 \le i \le n\}$.

Step 2: Assign values to memory strands.

Merge(T_{input}, T_V)

$\Longrightarrow T_{\text{input}} = \{C_1VE_0, \overline{C_0C_1}, E_1A_iD_0D_1B_iC_0 | 1 \le i \le n\}$.

Annealing(T_{input})

$$\Longrightarrow T_{\text{input}} = \left\{ \begin{bmatrix} E_1A_iD_0D_1B_iC_0C_1VE_0 \\ \overline{C_0C_1} \end{bmatrix}, 1 \le i \le n \right\}.$$

Denaturation(T_{input})

$\Longrightarrow T_{\text{input}} = \{\overline{C_0C_1}, E_1A_iD_0D_1B_iC_0C_1VE_0 | 1 \le i \le n\}$.

Separation(T_{input}, $\{\overline{C_0C_1}\}$, $T_{\bar{C}}$)

$\Longrightarrow T_{\text{input}} = \{E_1A_iD_0D_1B_iC_0C_1VE_0 | 1 \le i \le n\}$.

Copy(T_{input}, T_{output})

$\Longrightarrow T_{\text{output}} = \{E_1A_iD_0D_1B_iC_0C_1VE_0 | 1 \le i \le n\}$.

3. Procedure for F with DNA molecules

Let F be defined by (3). In this section we design a procedure with DNA molecules to check if there exists a $k \in \mathbb{N}$ such that $F^k(V^{(0)}) = (1, 1, \ldots, 1)$ for any given non-zero sequence $V^{(0)} = (V_1^{(0)}, V_2^{(0)}, \ldots, V_n^{(0)}) \in \{0, 1\}^n$. It is also

available for the general case $F^k(V^{(0)}) = \beta$ with $\beta \in \{0, 1\}^n$ after some minor modifications. Let

$$T_{\text{input}} = \{S_i, i = 1, \ldots, n\} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(0)} E_0, i = 1, \ldots, n\}$$

and

$$T_{\text{input}*} = \{D_1\#, \overline{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(0)} E_0 D_1\#}, i = 1, \ldots, n\}.$$

For $k \in \mathbb{N}$ let $V^{(k)} = F^k(V^{(0)}) := (V_1^{(k)}, V_2^{(k)}, \ldots, V_n^{(k)})$, the kth iteration of F on $V^{(0)}$. The test tubes T_{hitter} and T_{target} are used to store the first and second bits of a binary number, respectively.

For $k = 1$ to $k = 2^n - 1$

Step 1: Evaluate $V^{(k)} = F(V^{(k-1)}) = F^k(V^{(0)})$. Note that $V^{(k)} = F(V^{(k-1)}) = F_1 \circ F_2(V^{(k-1)})$. F_2 is evaluated by the first sub-steps [1-1]–[1-9]. F_1 is then evaluated by performing the operation of CycleShift in the sub-step [1-11].

[1-1] Separation(T_{input}, $\{B_1\}$, T_{hitter})
$\Longrightarrow T_{\text{input}} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k-1)} E_0 | 2 \le i \le n\}$.
[1-2] Separation(T_{input}, $\{B_2\}$, T_{target})
$\Longrightarrow T_{\text{input}} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k-1)} E_0 | 3 \le i \le n\}$.
[1-3] Copy(T_{hitter}, T_{tmp})
[1-4] Merge(T_{input}, T_{tmp})
$\Longrightarrow T_{\text{input}} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k-1)} E_0 | i \ne 2, 1 \le i \le n\}$.
[1-5] Copy(T_{target}, T_{tmp})
[1-6] Merge(T_{hitter}, T_{tmp})
$\Longrightarrow T_{\text{hitter}} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k-1)} E_0 | i = 1, 2\}$.
[1-7] Separation(T_{hitter}, $C_1 1 E_0$, T_{tmp})
[1-8] if Detect(T_{hitter}) is "no", then ValueAssignment$_0$(T_{target}, T_1); else if Detect(T_{tmp}) is "no", then ValueAssignment$_0$(T_{target}, T_1) else ValueAssignment$_1$(T_{target}, T_1)
$\Longrightarrow T_1 = \{E_1 A_2 D_0 D_1 B_2 C_0 C_1 \gamma(V_1^{(k-1)}, V_2^{(k-1)}) E_0\}$.
[1-9] Merge(T_{input}, T_1)
$\Longrightarrow T_{\text{input}} = \{E_1 A_2 D_0 D_1 B_2 C_0 C_1 \gamma(V_1^{(k-1)}, V_2^{(k-1)}) E_0, E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k-1)} E_0, i \ne 2, 1 \le i \le n\}$.
[1-10] Discard(T_{hitter}), (T_{target}) and (T_{tmp})
[1-11] CycleShift(T_{input}, T_{output})
$\Longrightarrow T_{\text{output}} = \{E_1 A_1 D_0 D_1 B_1 C_0 C_1 \gamma(V_1^{(k-1)}, V_2^{(k-1)}) E_0,$
$E_1 A_n D_0 D_1 B_n C_0 C_1 V_1^{(k-1)} E_0, E_1 A_{i-1} D_0 D_1 B_{i-1} C_0 C_1 V_i^{(k-1)} E_0, 3 \le i \le n\}$.
[1-12] Discard(T_{input})
[1-13] Merge(T_{input}, T_{output})
$\Longrightarrow T_{\text{input}} = \{E_1 A_1 D_0 D_1 B_1 C_0 C_1 \gamma(V_1^{(k-1)}, V_2^{(k-1)}) E_0,$
$E_1 A_n D_0 D_1 B_n C_0 C_1 V_1^{(k-1)} E_0, E_1 A_{i-1} D_0 D_1 B_{i-1} C_0 C_1 V_i^{(k-1)} E_0, 3 \le i \le n\} :=$
$\{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0, 1 \le i \le n\} = V^{(k)} = F^k(V^{(0)})$.

Step 2: Check whether or not $V^{(k)} = F^k(V^{(0)}) = (1, 1, \ldots, 1)$.

[2-1] Separation(T_{input}, $\{C_1 1 E_0\}$, T_{tmp})
$\Longrightarrow T_{\text{input}} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0 | V_i^{(k)} = 0, 1 \le i \le n\}$.
[2-2] if Detect(T_{input}) is "no", then end the procedure; else continue the following operations.
[2-3] Merge(T_{input}, T_{tmp})
$\Longrightarrow T_{\text{input}} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0 | 1 \le i \le n\}$.

Step 3: Check whether or not $V^{(k)} = F^k(V^{(0)}) = V^{(0)}$. If so, then $V^{(0)}$ is a periodic point of F and so there does not exist any $k \in \mathbb{N}$ such that $F^k(V^{(0)}) = (1, 1, \ldots, 1)$.

[3-1] Copy(T_{input}, T_{sk})
$\Longrightarrow T_{\text{sk}} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0 | 1 \le i \le n\}$

[3-2] Copy($T_{\text{input}*}$, T_{tmp})
$\Longrightarrow T_{\text{tmp}} = \{D_1\#, \overline{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(0)} E_0 D_1\#} | i = 1, \ldots, n\}$.

[3-3] Merge(T_{sk}, T_{tmp})
$\Longrightarrow T_{\text{sk}} = \{D_1\#, E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0, \overline{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(0)} E_0 D_1\#}, i = 1, \ldots, n\}$.

[3-4] Annealing(T_{sk})

$$\Longrightarrow T_{\text{sk}} = \left\{ \left[\frac{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0 D_1\#}{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(0)} E_0 D_1\#} \right] \middle| V_i^{(0)} = V_i^{(k)}, 1 \le i \le n \right\}$$

$$\bigcup \left\{ \left[\frac{D_1\#}{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(0)} E_0 D_1\#} \right] \middle| V_i^{(0)} \ne V_i^{(k)}, 1 \le i \le n \right\}$$

$\bigcup \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0 | V_i^{(0)} \ne V_i^{(k)}, 1 \le i \le n\}$.

[3-5] Denaturation(T_{sk})

[3-6] Separation(T_{sk}, $\{\bar{\#}\}$, $T_{\text{input}*}$)
$\Longrightarrow T_{\text{sk}} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0 D_1\# | V_i^{(0)} = V_i^{(k)}, 1 \le i \le n\}$
$\cup \{D_1\#, E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0 | V_i^{(0)} \ne V_i^{(k)}, 1 \le i \le n\}$.

[3-7] Separation(T_{sk}, $\{\#\}$, T_{tmp})
$\Longrightarrow T_{\text{sk}} = \{E_1 A_i D_0 D_1 B_i C_0 C_1 V_i^{(k)} E_0 | V_i^{(0)} \ne V_i^{(k)}, 1 \le i \le n\}$.

[3-8] if Detect(T_{sk}) is "no", then end the procedure ($V^{(0)}$ is a periodic point of F and there doesn't exist any $k \in \mathbb{N}$ such that $F^k(V^{(0)}) = (1, 1, \ldots, 1)$), else

[3-9] Discard(T_{sk}) and (T_{tmp})
End For

4. Conclusions

In 1994, Adleman (1994) published a pioneering work to show that DNA can be used to perform mathematical problems. Since then, lots of papers on DNA computing have been published. They are mainly involved in NP problems (e.g. Adleman, 1994; Lipton, 1995; Ouyang et al., 1997) and primitive operations, such as logic or arithmetic operations (e.g. Frisco, 2002; Fujiwara et al., 2004; Guarnieri et al., 1996; Gupta et al., 1997; Hug and Schuler, 2001; Qiu and Lu, 1998). The key to solve a mathematical problem using DNA molecules is to design appropriate DNA representations according to the problem itself. Fujiwara et al. (2004) designed a canonical DNA representation of a binary number (or an element in $\{0, 1\}^n$). Based on this representation they proposed procedures for logic and arithmetic operations by means of the basic DNA operations listed in Section 1. In this paper, we improve the DNA representation given by Fujiwara et al. (2004) for a binary number so that it is more able to deal with some mathematical problems. All implementations described in (Fujiwara et al., 2004) remain effective with this improved DNA representation. Moreover, this DNA representation can be used to make a procedure which can perform cycling shifts of elements in $\{0, 1\}^n$ in O(1) biological steps, so that one can study the dynamic behavior of some operators on $\{0, 1\}^n$ using DNA molecules. Once again it represents an evidence for the ability of DNA-based computing to solve mathematical problems. All our results in this paper are based on a theoretical model and the proposed procedures can be implemented practically since every DNA manipulation used in this model has been already realized in a lab.

As we know, there are still some problems currently for DNA computing experiments. One is the time involved in extracting and recombining DNA. While DNA processes within the test-tube can take place millions of times per second, extraction processes, whereby individual strands of DNA are manually isolated and spliced, can take several hours and even days, just for the simplest problems. This has led several researchers (e.g. Amos et al., 1997) to conclude that the complexity aspects of DNA algorithms will limit their applicability. The other is the error rate in the biological

manipulations (e.g. Li et al., 2003). However, we believe that these problems will be overcome as biological technique will be continually developed in the future.

Acknowledgements

The authors would like to thank the anonymous referees for their valuable comments and suggestions that lead to the improvement of the manuscript. This project is supported by Bio-X DNA Computer Consortium No. 03DZ14025. The second author is also supported by the National Science Foundation of China #10371043 and Shanghai Priority Academic Discipline.

References

Adleman, L.M., 1994. Molecular computation of solution to combinatorial problems. Science 266, 1021–1024.

Amos, M., Gibbons, A., Dunne, P.E., 1997. The complexity and viability of DNA computations. In: Lundh, D., Olsson, B., Narayanan, A. (Eds.), Biocomputing and Emergent Computation. World Scientific Press, pp. 165–173.

Frisco, P., 2002. Parallel arithmetic with splicing. Romanian J. Inform. Sci. Technol. 2, 113–128.

Fujiwara, A., Matsumoto, K., Chen, W., 2004. Procedures for logic and arithmetic operations with DNA molecules. Int. J. Found. Comput. Sci. 15, 461–474.

Guarnieri, F., Fliss, M., Bancroft, C., 1996. Making DNA add. Science 273, 220–223.

Gupta, V., Parthasarathy, S., Zaki, M.J., 1997. Arithmetic and logic operations with DNA. In: Proceedings of the Third DIMACS Workshop on DNA Based Computers, pp. 212–220.

Hug, H., Schuler, R., 2001. DNA-based parallel computation of simple arithmetic. In: Proceedings of the Seventh International Meeting on DNA Based Computers, pp. 159–166.

Li, D., Huang, H., Li, X., Li, X., 2003. Hairpin formation in DNA computation presents limits for large NP-complete problems. BioSyatems 72, 203–207.

Lipton, R.J., 1995. DNA solution of HARD computational problems. Science 268, 542–545.

Ouyang, Q., Kaplan, Peter, D., Liu, S., Libchaber, A., 1997. DNA solution of the maximal clique problem. Science 278, 446–449.

Păun, G., Rozeberg, G., Salomaa, A., 1998. DNA Computing. Springer-Verlag.

Qiu, Z.F., Lu, M., 1998. Arithmetic and logic operations for DNA computers. In: Proceedings of the Second IASTED International Conference on Parallel and Distributed Computing and Networks, pp. 481–486.