# Molecular Solutions For The Set-Partition Problem On Dna-Based Computing

Sientang Tsai and Wei-Yeh Chen

Department of Information Management, Southern Taiwan University of Science and Technology, Yuan Kung District, Tainan City, Taiwan, R.O.C.

## *ABSTRACT*

*Consider that the every element in a finite set S having q elements is a positive integer. The set-partition problem is to determine whether there is a subset $T \subseteq S$ such that $\sum_{x \in T} x = \sum_{x \in \overline{T}} x$, where $\overline{T}$ = {x| x ∈ S and x ∉ T}. This research demonstrates that molecular operations can be applied to solve the set-partition problem. In order to perform this goal, we offer two DNA-based algorithms, an unsigned parallel adder and a parallel Exclusive-OR (XOR) operation, that formally demonstrate our designed molecular solutions for solving the set-partition problem.*

## *KEYWORDS*

*DNA-based Computing, the Set-Partition Problem, the NP-Complete Problems, the NP-Hard Problems.*

## 1. Introduction

Feynman first offered bio-molecular computation in 1961, but his idea was not implemented by experiments for a few decades [16]. After almost thirty years later, Adleman finally achieved his experiment of the Hamiltonian path problem by manipulating DNA strands in a test tube [1]. From [19], that an optimal solution of every NP-complete or NP-hard problem is determined from its characteristics is manifested. DNA-based algorithms had been proposed to solve many computational problems, and those consisted of the satisfiability problem [21], the maximal clique [23], three-vertex-coloring [5], the subset-sum problem [11], the maximum cut problem [29], and the binary integer programming problem [30]. One potentially significant area of application for DNA algorithms is the breaking of encryption schemes [9]. From [18], DNA-based arithmetic algorithms are proposed, and from [25] DNA-based algorithms for constructing DNA databases are also offered. Here we use the molecular operations in the Adleman-Lipton filtering model to develop the DNA-based algorithms for solving the set-partition problem. We also construct an unsigned parallel adder and a parallel Exclusive-OR (XOR) operation in the procedure of demonstration.

## 2. DNA Computation

In this section we describe the available techniques for dealing with DNA strands that will be used to solve the set-partition problem.

DOI : 10.5121/ijcsit.2013.5602 

### 2.1. Descriptions of Molecular Operations in DNA Manipulations

There have been revolutionary advances in the field of biomedical engineering particularly in recombinant DNA and RNA manipulating in the last decade. Due to the industrialization of the biotechnology field, laboratory techniques for recombinant DNA and RNA manipulation are becoming highly standardized. Basic principles about recombinant DNA can be found in [6, 7, 15, 28]. In this subsection we describe eight biological operations useful for solving the set-partition problem. The method of constructing DNA solution space for the set-partition problem is based on the proposed method in [3, 10].

The filtering model developed by Adleman is memory less in the sense that the strings do not change during a computation. A computation composes of a sequence of operations on finite multi-set of strings containing the alphabet $\{A, C, G, T\}$. A test tube is a finite multi-set of strings. This work is based on the following operations:

1. *Extract*: Consider a test tube $P$ and a short DNA substring $x$, create two new tubes $+(P, x)$ and $-(P, x)$, where $+(P, x)$ consists of all strands in $P$ containing $x$ as a sequence, while $-(P, x)$ produces all strands in $P$ not containing $x$ *as a* sequence.
2. *Merge*: Take two test tubes $P_1$ and $P_2$, produce their union $P_1 \cup P_2$, and place the result into the tube $\cup(P_1, P_2)$, where $\cup(P_1, P_2) = P_1 \cup P_2$.
3. *Detect:* Pick a test tube $P$ and output 'yes' if $P$ contains at least one DNA molecule; otherwise output 'no'.
4. *Amplify*: Start with a test tube $P$, this operation, *Amplify* $(P, P_1, P_2)$, will duplicate two new copies $P_1$ and $P_2$ of $P$ and $P$ becomes an empty tube.
5. *Append*: Take a test tube $P$ and a short strand $x$, this operation will create a tube that contains all strands having string $x$ at the end of every strand in $P$.
6. *Append-head*: Consider a test tube $P$ and a short strand $x$, and generate a test tube that comprises all strands having string $x$ at the beginning of every strand in $P$.
7. *Read*: this operation describes each of the resulting solutions contained in tube $P$. If $P$ is empty, then no answer is found.

## 3. DNA Algorithms for Solving the Set-Partition Problem

### 3.1. Definition of the Set-Partition Problem

Assume that a finite set $S$ is $\{s_1, s_2, \ldots, s_q\}$, where $s_m$ is the $m$th element for $1 \le m \le q$. Also consider every element in $S$ is a positive integer. The set-partition problem is to decide if there is a subset $T \subseteq S$ such that $\sum_{x \in T} x = \sum_{x \in \bar{T}} x$, where $\bar{T} = \{x \mid x \in S \text{ and } x \notin T\}$. The set-partition problem has been verified to be the NP-complete problem [13, 17].

Consider that a finite set $S = \{1, 2, 3\}$. The complete subsets for $S$ are $\varnothing$, $\{1\}$, $\{2\}$, $\{3\}$, $\{1, 2\}$, $\{2, 3\}$, $\{1, 3\}$ and $\{1, 2, 3\}$ respectively. According to the definition above of the set-partition problem, two subsets $T$ and subset $\bar{T}$ are disjoint, i.e. $T \cap \bar{T} = \varnothing$, and they form $S$, i.e. $T \cup \bar{T} = S$. Therefore, the set $S$ has four partitions: (1) $T = \{1,3\}$ and $\bar{T} = \{2\}$; (2) $T = \{2,3\}$ and $\bar{T} = \{1\}$; (3) $T = \{3\}$ and $\bar{T} = \{1, 2\}$; (4) $T = \{1, 2, 3\}$ and $\bar{T} = \varnothing$. Subsequently, the sum for each pair $(T, \bar{T})$ is

(4, 2), (5, 1), (3, 3) and (6, 0). According to the definition of Set-Partition Problem, the solution for a finite set S is T = {3} and $\bar{T}$ = {1, 2}.

### 3.2. Creating the Solution Space of DNA Strands

Suppose that $x_q x_{q-1} \ldots x_2 x_1$ is a $q$-bit binary number applied to represent $q$ elements in a finite set $S$, where the value of each bit $x_m$ is either 1 or 0 for $1 \le m \le q$. From [3, 10], for every bit $x_m$ representing the $m$th element in $S$ for $1 \le m \le q$, two distinct 15-base value sequences are designed. One represents the value "0" for $x_m$ and the other represents the value "1" for $x_m$. For sake of convenience in our representation, assume that $x_m^1$, which corresponding element $s_m$ belongs to the subset of $S$, denotes the value of $x_m$ to be 1 and $x_m^0$, which corresponding element $s_m$ does *not* belong to the subset of $S$, denotes the value of $x_m$ to be 0. The following DNA-based algorithm is used to construct solution the space of DNA strands for solving the set-partition problem of a $q$-element set $S$.

**Procedure Init** ($T_0$, $q$)

(1) Append($T_1$, $x_{q-1}^1$)

(2) Append($T_2$, $x_{q-1}^0$)

(3) $T_0 = \cup(T_1, T_2)$

(4) **For** $m = q - 2$ **downto** 1

(4a) Amplify($T_0$, $T_1$, $T_2$)

(4b) Append($T_1$, $x_m^1$)

(4c) Append($T_2$, $x_m^0$)

(4d) $T_0 = \cup(T_1, T_2)$

**EndFor**

(5) Append-head($T_0$, $x_q^1$)

**EndProcedure**

Consider that a finite set $S$ is equal to {001, 010, 011}. The number of elements in the finite set $S$, $q$ is three. When the algorithm, **Init** ($T_0$, $q$), is called from step (1) of **Algorithm 1** in Subsection 3.8, it is applied to construct the solution space of DNA strands. Tube $T_0$ is an empty tube. We think of it as an input tube for **Init** ($T_0$, $q$) and the number of elements in the finite set $S$, $q$, as the second parameter for **Init** ($T_0$, $q$). After the execution of Step(1) and step (2) are performed, tube $T_1 = \{ x_2^1 \}$ and tube $T_2 = \{ x_2^0 \}$. Then, the *merge* operation makes tube $T_0 = \{ x_2^1, x_2^0 \}$, tube $T_1 = \varnothing$, and tube $T_2 = \varnothing$ after step (3).

Because the value of $q$ is three, step (4a) through (4d) will be executed one time. We perform *Amplify* operation in step (4a), then tube $T_0 = \varnothing$, tube $T_1 = \{ x_2^1, x_2^0 \}$, and tube $T_2 = \{ x_2^1, x_2^0 \}$. Next, after the execution of step (4b) and the execution of step (4c) are performed, tube $T_1 = \{ x_2^1 x_1^1, x_2^0 x_1^1 \}$ and tube $T_2 = \{ x_2^1 x_1^0, x_2^0 x_1^0 \}$. Then, after having finished step (4d), the *merge* operation makes tube $T_0 = \{ x_2^1 x_1^1, x_2^0 x_1^1, x_2^1 x_1^0, x_2^0 x_1^0 \}$, tube $T_1 = \varnothing$ and tube $T_2 = \varnothing$. Finally, after the execution of step (5) is done, tube $T_0 = \{ x_3^1 x_2^1 x_1^1, x_3^1 x_2^0 x_1^1, x_3^1 x_2^1 x_1^0, x_3^1 x_2^0 x_1^0 \}$.

The result is shown in Table 1. Lemma 1 is used to demonstrate the correctness of the algorithm **Init** ($T_0$, $q$).

Table 1. **Init**($T_0$, $q$) procedure produces the following results.

| Tube | The result generated by **Init**($T_0$, $q$) |
|---|---|
| $T_0$ | $\{ x_3^1 x_2^1 x_1^1,\ x_3^1 x_2^0 x_1^1,\ x_3^1 x_2^1 x_1^0,\ x_3^1 x_2^0 x_1^0 \}$ |
| $T_1$ | $\varnothing$ |
| $T_2$ | $\varnothing$ |

**Lemma 1:** The **Init** ($T_0$, $q$) procedure creates the solution space of DNA strands for a $q$-element set $S$.

**Proof:** The **Init** ($T_0$, $q$) procedure is implemented by means of *amplify, append*, *append-head* and *merge* operations. Step (1) and (2) append the DNA sequences representing value "1" for $x_{q-1}$ and value "0" for $x_{q-1}$, respectively, onto the end of every strand in tube $T_1$ and $T_2$. Hence subsets including the $(q-1)$th element appear in $T_1$ and subsets not including the $(q-1)$th element appear in $T_2$. After step (3), tube $T_0$ is merged by tube $T_1$ and $T_2$. This means that DNA strands in $T_0$ include sequences of $x_{q-1} = 1$ and $x_{q-1} = 0$.

Then, each time step (4a) is performed, it uses the *amplify* operation to copy the content of tube $T_0$ into two new tubes, $T_1$ and $T_2$, which are copies of $T_0$. Tube $T_0$ becomes empty. Step(4b) and step (4c) are used to subsequently append DNA sequences, respectively, representing the value "1" or "0" for $x_m$, onto the end of every strand in tube $T_1$ and tube $T_2$. This implies that subsets containing the $m$th element appear in tube $T_1$ and subsets not containing the $m$th element appear in tube $T_2$. Finally, tube $T_0$ is merged by tube $T_1$ and $T_2$ in step (4d). This indicates that DNA strands in tube $T_0$ include DNA sequences of $x_m = 1$ and $x_m = 0$. After For loop in step 4 is performed completely, $T_0$ is comprised of $2^{q-1}$ DNA sequences. After step (5) is finished, a DNA sequence, representing the value "1" for $x_q$, is appended onto the head of every strand in tube $T_0$. We conclude that this procedure can create $2^{q-1}$ partitions of a $q$-element set $S$ with DNA strands.

### 3.3. Solution Space of the Value for Every Element of Each Subset for Solving the Set-Partition Problem of a Finite Set

For the sake of designing a better and simpler DNA-based algorithm for solving the set-partition problem of a $q$-element finite set $S$, suppose that for an element, $s_m \in T$, its value is represented as a binary number of $n$ bits, $s_{m,n}, s_{m,n-1}, \ldots, s_{m,2}, s_{m,1}$ and for an element, $r_m \in \bar{T}$, its value is represented as a binary number of $n$ bits, $r_{m,n}, r_{m,n-1}, \ldots, r_{m,2}, r_{m,1}$. Also suppose that $s_{m,n}$ and $r_{m,n}$ are the most significant bit, and $s_{m,1}$ and $r_{m,1}$ are the least significant bit. For every bit $s_{m,k}$ and $r_{m,k}$, $1 \le m \le q$ and $1 \le k \le n$, from [3, 10] two *distinct* DNA sequences are designed. One corresponds to the value "0" for $s_{m,k}$ and $r_{m,k}$ and the other corresponds to the value "1" for $s_{m,k}$ and $r_{m,k}$. For the sake of convenience in our representation, assume that $s_{m,k}^1$ and $r_{m,k}^1$ denote the value of $s_{m,k}$ and $r_{m,k}$ to be 1 and $s_{m,k}^0$ and $r_{m,k}^0$ define the value of $s_{m,k}$ and $r_{m,k}$ to be 0. The following algorithm is employed to construct the value of each corresponding element in $2^{q-1}$ partitions of a $q$-element set $S$.

**Procedure Value**($T_0$, $q$, $n$)

(1) **For** $m = 1$ **to** $q$

(1a) $T_1 = +(T_0, x_m^1)$ and $T_2 = -(T_0, x_m^1)$

(1b) **For** $k = n$ **downto** 1

(1c) Append ($T_1$, $s_{m,k}$)

(1d) Append ($T_2$, $s_{m,k}^0$)

**End For**

(1e) $T_0 = \cup(T_1, T_2)$

**EndFor**

(2) **For** $m = 1$ **to** $q$

(2a) $T_1 = +(T_0, x_m^1)$ and $T_2 = -(T_0, x_m^1)$

(2b) **For** $k = n$ **downto** 1

(2c) Append ($T_1$, $r_{m,k}^0$)

(2d) Append ($T_2$, $r_{m,k}$)

**End For**

(2e) $T_0 = \cup(T_1, T_2)$

**EndFor**

**EndProcedure**

When the algorithm, **Value**($T_0$, $q$, $n$), is called from step (2) of **Algorithm 1** in Subsection 3.8, it is used to encode the value of each element in $2^{q-1}$ partitions of a $q$-element set $S$. We think of tube $T_0$ with the result shown in Table 1 as an input tube for the algorithm, **Value**($T_0$, $q$, $n$), the number of elements in $S$, $q$ is regarded as the second parameter, and the number of bits for each element, $n$, is regarded as the third parameter. Step(1) is the first nested loop and is applied to construct the value of each element in $T$ in each pair of $(T, \bar{T})$. After we perform step (1a), then tube $T_1 = \{x_3^1 x_2^1 x_1^1, x_3^1 x_2^0 x_1^1\}$ and tube $T_2 = \{x_3^1 x_2^1 x_1^0, x_3^1 x_2^0 x_1^0\}$. Then, after the first execution of step (1c) and (1d) is performed, tube $T_1 = \{x_3^1 x_2^1 x_1^1 s_{1,3}^0, x_3^1 x_2^0 x_1^1 s_{1,3}^0\}$ and tube $T_2 = \{x_3^1 x_2^1 x_1^0 s_{1,3}^0, x_3^1 x_2^0 x_1^0 s_{1,3}^0\}$. After every execution for step (1c) and (1d) is finished, tube $T_1 = \{x_3^1 x_2^1 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1, x_3^1 x_2^0 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1\}$ and tube $T_2 = \{x_3^1 x_2^1 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0, x_3^1 x_2^0 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0\}$. Then, after the first execution of step (1e) is finished, tube $T_0 = \{x_3^1 x_2^1 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1, x_3^1 x_2^0 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1, x_3^1 x_2^1 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0, x_3^1 x_2^0 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0\}$, tube $T_1 = \varnothing$ and tube $T_2 = \varnothing$. Next, after each execution of step (1a) through (1e) in the first nested loop is completed, tube $T_0 = \{x_3^1 x_2^1 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, x_3^1 x_2^1 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, x_3^1 x_2^0 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1 s_{2,3}^0 s_{2,2}^0 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, x_3^1 x_2^0 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^0 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1\}$, tube $T_1 = \varnothing$ and tube $T_2 = \varnothing$. Step(2) is the second nested loop and is used to construct the value of each element in $\bar{T}$ in each pair of $(T, \bar{T})$. The contents of tube $T_0$ are shown as above. When the first execution of step (2a) is performed, tube $T_1 = \{x_3^1 x_2^1 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, x_3^1 x_2^0 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1 s_{2,3}^0 s_{2,2}^0 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1\}$ and tube $T_2 = \{x_3^1 x_2^1 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, x_3^1 x_2^0 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^0 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1\}$. Then, after the first execution, the second execution and the third execution for step (2c) and (2d) are finished, "$r_{1,3}^0 r_{1,2}^0 r_{1,1}^0$" is appended onto the tail of each bit pattern in tube $T_1$, and "$r_{1,3}^0 r_{1,2}^0 r_{1,1}^1$" is appended onto the tail of each bit pattern in tube $T_2$. Next, after the first execution of step (2e) is performed, tube $T_0 = \{x_3^1 x_2^1 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1 r_{1,3}^0 r_{1,2}^0 r_{1,1}^0, x_3^1 x_2^1 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, r_{1,3}^0 r_{1,2}^0 r_{1,1}^1, x_3^1 x_2^0 x_1^1 s_{1,3}^0 s_{1,2}^0$

$s_{1,1}^{1}\ s_{2,3}^{0}\ s_{2,2}^{0}\ s_{2,1}^{0}\ s_{3,3}^{0}\ s_{3,2}^{1}\ s_{3,1}^{1}$, $r_{1,3}^{0}\ r_{1,2}^{0}\ r_{1,1}^{0}$, $x_3^{1}\ x_2^{0}\ x_1^{0} s_{1,3}^{0}\ s_{1,2}^{0}\ s_{1,1}^{0}\ s_{2,3}^{0}\ s_{2,2}^{0}\ s_{2,1}^{0}\ s_{3,3}^{0}\ s_{3,2}^{1}\ s_{3,1}^{1}\ r_{1,3}^{0}$ $r_{1,2}^{0}\ r_{1,1}^{1}\}$, tube $T_1 = \varnothing$ and tube $T_2 = \varnothing$. Then, after the second For loop is done, tube $T_1 = \varnothing$, tube $T_2 = \varnothing$ and the contents of tube $T_0$ are shown in Table 2. Lemma 2 is applied to prove the correctness of the algorithm, **Value** ($T_0$, $q$, $n$).

Table 2. The result generated by **Value** ($T_0$, $q$, $n$).

| Tube | The result generated by **Value**($T_0$, $q$, $n$) |
|---|---|
| $T_0$ | $\{x_3^{1}\ x_2^{1}\ x_1^{1}\ s_{1,3}^{0}\ s_{1,2}^{0}\ s_{1,1}^{1}\ s_{2,3}^{0}\ s_{2,2}^{1}\ s_{2,1}^{0}\ s_{3,3}^{0}\ s_{3,2}^{1}\ s_{3,1}^{1}\ r_{1,3}^{0}\ r_{1,2}^{0}\ r_{1,1}^{0}\ r_{2,3}^{0}\ r_{2,2}^{0}\ r_{2,1}^{0}\ r_{3,3}^{0}\ r_{3,2}^{0}\ r_{3,1}^{0}$,<br>$x_3^{1}\ x_2^{1}\ x_1^{0}\ s_{1,3}^{0}\ s_{1,2}^{0}\ s_{1,1}^{0}\ s_{2,3}^{0}\ s_{2,2}^{1}\ s_{2,1}^{0}\ s_{3,3}^{0}\ s_{3,2}^{1}\ s_{3,1}^{1}$, $r_{1,3}^{0}\ r_{1,2}^{0}\ r_{1,1}^{1}\ r_{2,3}^{0}\ r_{2,2}^{0}\ r_{2,1}^{0}\ r_{3,3}^{0}\ r_{3,2}^{0}\ r_{3,1}^{0}$,<br>$x_3^{1}\ x_2^{0}\ x_1^{1}\ s_{1,3}^{0}\ s_{1,2}^{0}\ s_{1,1}^{1}\ s_{2,3}^{0}\ s_{2,2}^{0}\ s_{2,1}^{0}\ s_{3,3}^{0}\ s_{3,2}^{1}\ s_{3,1}^{1}$, $r_{1,3}^{0}\ r_{1,2}^{0}\ r_{1,1}^{0}\ r_{2,3}^{0}\ r_{2,2}^{1}\ r_{2,1}^{0}\ r_{3,3}^{0}\ r_{3,2}^{0}\ r_{3,1}^{0}$,<br>$x_3^{1}\ x_2^{0}\ x_1^{0} s_{1,3}^{0}\ s_{1,2}^{0}\ s_{1,1}^{0}\ s_{2,3}^{0}\ s_{2,2}^{0}\ s_{2,1}^{0}\ s_{3,3}^{0}\ s_{3,2}^{1}\ s_{3,1}^{1}\ r_{1,3}^{0}\ r_{1,2}^{0}\ r_{1,1}^{1}\ r_{2,3}^{0}\ r_{2,2}^{1}\ r_{2,1}^{0}\ r_{3,3}^{0}$ $r_{3,2}^{0}\ r_{3,1}^{0}\}$ |

**Lemma 2:** The value of each element in each pair of $(T,\bar{T})$ for a $q$-element finite set, $S$, can be constructed from the algorithm, **Value** ($T_0$, $q$, $n$).
**Proof**: Refer to Lemma 1.

## 3.4. The Implementation of a Parallel One-bit Adder

A one-bit adder has three inputs: the two data input and a carry input. It forms the arithmetic sum of these inputs. Two of the input bits represent two significant bits to be added. The third input represents the carry from the previous lower significant position. The least significant bit of the sum for augend, addend and previous carry comes from first output. The output carry transferred into the input carry of the next one-bit adder comes from second output. The truth table of a one-bit adder is as follows:

Table 3. The truth table of a one-bit adder.

| Augend bit | Addend bit | Previous carry bit | Sum bit | Carry bit |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

Suppose that two one-bit binary numbers, $\alpha_{m-1,\,k}$ and $\alpha_{m,\,k}$, represent the first input (augend) and the first output (sum) of a one-bit adder for $1 \le m \le q$ and $1 \le k \le n$, respectively. A one-bit binary number, $\beta_{m,\,k}$, is applied to represent the second input (addend) of a one-bit adder. $\gamma_{m,\,k-1}$ is used to represent the third input (previous carry) and $\gamma_{m,\,k}$ represent the second output (current carry) of a

one-bit adder. From [3, 10], two *distinct* DNA sequences are designed to represent the value "0" and "1" for every corresponding bit. Also suppose that $\beta^{1}_{m,k}$ contains the value of $\beta_{m,k}$ to be 1 and $\beta^{0}_{m,k}$ contains the value of $\beta_{m,k}$ to be 0. Also suppose that $\alpha^{1}_{m-1,k}$ denotes the value of $\alpha_{m\text{-}1,k}$ to be 1 and $\alpha^{0}_{m-1,k}$ defines the value of $\alpha_{m\text{-}1,k}$ to be 0. Similarly, suppose that $\alpha^{1}_{m,k}$ contains the value of $\alpha_{m,k}$ to be 1 and $\alpha^{0}_{m,k}$ denotes the value of $\alpha_{m,k}$ to be 0, $\gamma^{1}_{m,k-1}$ denotes the value of $\gamma_{m,k\text{-}1}$ to be 1 and $\gamma^{0}_{m,k-1}$ contains the value of $\gamma_{m,k\text{-}1}$ to be 0. $\gamma_{m,k}{}^{1}$ defines the value of $\gamma_{m,k}$ to be 1 and $\gamma_{m,k}{}^{0}$ contains the value of $\gamma_{m,k}$ to be 0. The following algorithm is offered to perform the Boolean function of a parallel one-bit adder.

**Procedure ParallelOneBitAdder**($T_0$, $\alpha_{m\text{-}1,k}$, $\beta_{m,k}$, $\gamma_{m,k\text{-}1}$, $m$, $k$)

(1) $T_1 = +(T_0, \alpha^{1}_{m-1,k})$ and $T_2 = -(T_0, \alpha^{1}_{m-1,k})$

(2) $T_3 = +(T_1, \beta^{1}_{m,k})$ and $T_4 = -(T_1, \beta^{1}_{m,k})$

(3) $T_5 = +(T_2, \beta^{1}_{m,k})$ and $T_6 = -(T_2, \beta^{1}_{m,k})$

(4) $T_7 = +(T_3, \gamma^{1}_{m,k-1})$ and $T_8 = -(T_3, \gamma^{1}_{m,k-1})$

(5) $T_9 = +(T_4, \gamma^{1}_{m,k-1})$ and $T_{10} = -(T_4, \gamma^{1}_{m,k-1})$

(6) $T_{11} = +(T_5, \gamma^{1}_{m,k-1})$ and $T_{12} = -(T_5, \gamma^{1}_{m,k-1})$

(7) $T_{13} = +(T_6, \gamma^{1}_{m,k-1})$ and $T_{14} = -(T_6, \gamma^{1}_{m,k-1})$

(8) **If** (Detect ($T_7$) = = "yes") **then**

Append-head ($T_7$, $\alpha^{1}_{m,k}$) and Append-head ($T_7$, $\gamma^{1}_{m,k}$)

**EndIf**

(9) **If** (Detect ($T_8$) = = "yes") **then**

Append-head ($T_8$, $\alpha^{0}_{m,k}$) and Append-head ($T_8$, $\gamma^{1}_{m,k}$)

**EndIf**

(10) **If** (Detect ($T_9$) = = "yes") **then**

Append-head ($T_9$, $\alpha^{0}_{m,k}$) and Append-head ($T_9$, $\gamma^{1}_{m,k}$)

**EndIf**

(11) **If** (Detect ($T_{10}$) = = "yes") **then**

Append-head ($T_{10}$, $\alpha^{1}_{m,k}$) and Append-head ($T_{10}$, $\gamma^{0}_{m,k}$)

**EndIf**

(12) **If** (Detect ($T_{11}$) = = "yes") **then**

Append-head ($T_{11}$, $\alpha^{0}_{m,k}$) and Append-head ($T_{11}$, $\gamma^{1}_{m,k}$)

**EndIf**

(13) **If** (Detect ($T_{12}$) = ="yes") **then**

Append-head ($T_{12}$, $\alpha^{1}_{m,k}$) and Append-head ($T_{12}$, $\gamma^{0}_{m,k}$)

**EndIf**

(14) **If** (Detect ($T_{13}$) = = "yes") **then**

Append-head ($T_{13}, \alpha^{1}_{m,k}$) and Append-head ($T_{13}, \gamma^{0}_{m,k}$)

**EndIf**

(15) **If** (Detect ($T_{14}$) = = "yes") **then**

Append-head ($T_{14}, \alpha^{0}_{m,k}$) and Append-head ($T_{14}, \gamma^{0}_{m,k}$)

**EndIf**

(16) $T_0 = \cup(T_7, T_8, T_9, T_{10}, T_{11}, T_{12}, T_{13}, T_{14})$

**EndProcedure**

**Lemma 3:** The **ParallelOneBitAdder**($T_0$, $\alpha_{m-1,\,k}$, $\beta_{m,\,k}$, $\gamma_{m,\,k-1}$, $m$, $k$) procedure can be applied to implement a parallel one-bit adder.

**Proof:** The algorithm **ParallelOneBitAdder**($T_0$, $\alpha_{m-1,\,k}$, $\beta_{m,\,k}$, $\gamma_{m,\,k-1}$, $m$, $k$) is implemented via the *extract, append-head* and *merge* operations. Steps from (1) to (7) employ the *extract* operation to form some different test tubes including different strands ($T_1$ to $T_{14}$). That is $T_1$ includes all of the strands that have $\alpha_{m-1,\,k} = 1$, $T_2$ includes all of the strands that have $\alpha_{m-1,\,k} = 0$. $T_3$ includes all of the strands that have $\alpha_{m-1,\,k} = 1$ and $\beta_{m,\,k} = 1$. $T_4$ includes those that have $\alpha_{m-1,\,k} = 1$ and $\beta_{m,\,k} = 0$. $T_5$ includes those that have $\alpha_{m-1,\,k} = 0$ and $\beta_{m,\,k} = 1$. $T_6$ includes those that have $\alpha_{m-1,k} = 0$ and $\beta_{m,k} = 0$. $T_7$ includes those that have $\alpha_{m-1,k} = 1$, $\beta_{m,k} = 1$, and $\gamma_{m,k-1} = 1$. $T_8$ includes those that have $\alpha_{m-1,\,k} = 1$, $\beta_{m,k} = 1$, and $\gamma_{m,\,k-1} = 0$. $T_9$ includes those that have $\alpha_{m-1,\,k} = 1$, $\beta_{m,\,k} = 0$, and $\gamma_{m,\,k-1} = 1$. $T_{10}$ includes those that have $\alpha_{m-1,\,k} = 1$, $\beta_{m,\,k} = 0$, and $\gamma_{m,\,k-1} = 0$. $T_{11}$ includes those that have $\alpha_{m-1,\,k} = 0$, $\beta_{m,\,k} = 1$, and $\gamma_{m\,,\,k-1} = 1$. $T_{12}$ includes those that have $\alpha_{m-1,\,k} = 0$, $\beta_{m,\,k} = 1$, and $\gamma_{m,\,k-1} = 0$. $T_{13}$ includes those that have $\alpha_{m-1,\,k} = 0$, $\beta_{m,\,k} = 0$, and $\gamma_{m,\,k-1} = 1$. Finally $T_{14}$ consists of those that have $\alpha_{m-1,\,k} = 0$, $\beta_{m,\,k} = 0$, and $\gamma_{m,\,k-1} = 0$. After step (1) through step (7) are performed, those eight corresponding results of a one-bit adder as shown in Table 3 are poured into tube $T_7$ through $T_{14}$ respectively.

Next, we use step (8) through step (15) to check whether it contains any DNA strand for tubes $T_7$, $T_8$, $T_9$, $T_{10}$, $T_{11}$, $T_{12}$, $T_{13}$, and $T_{14}$ or not. If any a "yes" returned for those steps, then *append-head* operations will be performed correspondingly. The results are $\alpha^{1}_{m,k}$ or $\alpha^{0}_{m,k}$, and $\gamma^{1}_{m\,,\,k}$ or $\gamma^{0}_{m\,,\,k}$ are appended onto the head of every strand in the corresponding tubes. After performing step (8) through step (15), we find that one of eight different outputs of a one-bit adder in Table 3 is correspondingly appended into tubes $T_7$ through $T_{14}$. The last execution of step (16) applies the *merge* operation to pour tubes $T_7$ through $T_{14}$ into tube $T_0$. $T_0$ contains the strands performing the addition of three input bits.

## 3.5. The Implementation of a Parallel N-bit Adder

The parallel one-bit adder introduced in Section 3.4 figures out the arithmetic sum of two bits and a previous carry. A binary parallel *n*-bit adder also performs the arithmetic sum for the two input operands of *n*-bit and the input carry through performing this one-bit adder *n* times. The following algorithm is offered to perform the arithmetic sum for a parallel *n*-bit adder.

**Procedure ParallelAdder**($T_0$, $\alpha$, $\beta$, $\gamma$, $q$, $n$)

(1) **For** k = $n$ **downto** 1

(1a) Append ($T_0, \alpha^{0}_{0,k}$)

**EndFor**

(2) **For** $m = 1$ **to** $q$

(2a) Append $(T_0, \gamma_{m,0}^0)$

(2b) **For** $k = 1$ **to** $n$

(2c) **ParallelOneBitAdder**$(T_0, \alpha_{m\text{-}1,\,k}, \beta_{m,\,k}, \gamma_{m,\,k\text{-}1}, m, k)$

**EndFor**

**EndFor**

**EndProcedure**

When the first execution of **ParallelAdder**$(T_0, \alpha, \beta, \gamma, q, n)$ is called from step (3) of **Algorithm 1** in Section 3.8, tube $T_0$ with the result in Table 2 in Section 3.3 is regarded as the input tube. From the second parameter to the fourth parameter are replaced by the actual arguments $A$, $s$ and $z$, respectively. The values for the fifth and sixth parameters, $q$ and $n$, are both three. Because the value of $n$ is equal to three, three bits $A_{0,3}^0$, $A_{0,2}^0$ and $A_{0,1}^0$ are appended onto the tail of each bit pattern in tube $T_0$ after the execution of step (1a). Since the values for $n$ and $q$ are both three, step (2a) will be executed three times and step (2c) will be executed nine times. The bit, $z_{1,0}^0$, from the first execution of step (2a) is appended onto the tail of each bit pattern in tube $T_0$. Next, after the first execution, the second execution and the third execution for step (2c), that call the algorithm, **ParallelOneBitAdder**$(T_0, \alpha_{m\text{-}1,\,k}, \beta_{m,\,k}, \gamma_{m,\,k\text{-}1}, m, k)$, are performed, tube $T_0 = \{z_{1,3}^0 A_{1,3}^0 z_{1,2}^0 A_{1,2}^0 z_{1,1}^0 A_{1,1}^1 x_3^1 x_2^1 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1 r_{1,3}^0 r_{1,2}^0 r_{1,1}^0 r_{2,3}^0 r_{2,2}^0 r_{2,1}^0 r_{3,3}^0 r_{3,2}^0 r_{3,1}^0 A_{0,3}^0 A_{0,2}^0 A_{0,1}^0 z_{1,0}^0, z_{1,3}^0 A_{1,3}^0 z_{1,2}^0 A_{1,2}^0 z_{1,1}^0 A_{1,1}^0 x_3^1 x_2^1 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, r_{1,3}^0 r_{1,2}^0 r_{1,1}^1 r_{2,3}^0 r_{2,2}^0 r_{2,1}^0 r_{3,3}^0 r_{3,2}^0 r_{3,1}^0 A_{0,3}^0 A_{0,2}^0 A_{0,1}^0 z_{1,0}^0, z_{1,3}^0 A_{1,3}^0 z_{1,2}^0 A_{1,2}^0 z_{1,1}^0 A_{1,1}^1 x_3^1 x_2^0 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1 s_{2,3}^0 s_{2,2}^0 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, r_{1,3}^0 r_{1,2}^0 r_{1,1}^0 r_{2,3}^0 r_{2,2}^1 r_{2,1}^0 r_{3,3}^0 r_{3,2}^0 r_{3,1}^0, A_{0,3}^0 A_{0,2}^0 A_{0,1}^0 z_{1,0}^0, z_{1,3}^0 A_{1,3}^0 z_{1,2}^0 A_{1,2}^0 z_{1,1}^0 A_{1,1}^0 x_3^1 x_2^0 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^0 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1 r_{1,3}^0 r_{1,2}^0 r_{1,1}^1 r_{2,3}^0 r_{2,2}^1 r_{2,1}^0 r_{3,3}^0 r_{3,2}^0 r_{3,1}^0 A_{0,3}^0 A_{0,2}^0 A_{0,1}^0 z_{1,0}^0\}$. Similarly, after each operation in step (2) is performed, the result for tube $T_0$ is shown in Table 4.

Table 4. The first execution result generated by **ParallelAdder**$(T_0, \alpha, \beta, \gamma, q, n)$.

| Tube | The first execution result generated by **ParallelAdder**$(T_0, \alpha, \beta, \gamma, q, n)$ |
|---|---|
| $T_0$ | $\{z_{3,3}^0 A_{3,3}^1 z_{3,2}^1 A_{3,2}^1 z_{3,1}^1 A_{3,1}^0 z_{2,3}^0 A_{2,3}^0 z_{2,2}^0 A_{2,2}^1 z_{2,1}^0 A_{2,1}^1 z_{1,3}^0 A_{1,3}^0 z_{1,2}^0 A_{1,2}^0 z_{1,1}^0 A_{1,1}^1 x_3^1 x_2^1 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1 r_{1,3}^0 r_{1,2}^0 r_{1,1}^0 r_{2,3}^0 r_{2,2}^0 r_{2,1}^0 r_{3,3}^0 r_{3,2}^0 r_{3,1}^0 A_{0,3}^0 A_{0,2}^0 A_{0,1}^0 z_{1,0}^0 z_{2,0}^0 z_{3,0}^0,$ $z_{3,3}^0 A_{3,3}^1 z_{3,2}^1 A_{3,2}^0 z_{3,1}^1 A_{3,1}^1 z_{2,3}^0 A_{2,3}^0 z_{2,2}^0 A_{2,2}^1 z_{2,1}^0 A_{2,1}^0 z_{1,3}^0 A_{1,3}^0 z_{1,2}^0 A_{1,2}^0 z_{1,1}^0 A_{1,1}^0 x_3^1 x_2^1 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^1 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, r_{1,3}^0 r_{1,2}^0 r_{1,1}^1 r_{2,3}^0 r_{2,2}^0 r_{2,1}^0 r_{3,3}^0 r_{3,2}^0 r_{3,1}^0 A_{0,3}^0 A_{0,2}^0 A_{0,1}^0 z_{1,0}^0 z_{2,0}^0 z_{3,0}^0,$ $z_{3,3}^0 A_{3,3}^1 z_{3,2}^1 A_{3,2}^0 z_{3,1}^1 A_{3,1}^0 z_{2,3}^0 A_{2,3}^0 z_{2,2}^0 A_{2,2}^0 z_{2,1}^0 A_{2,1}^1 z_{1,3}^0 A_{1,3}^0 z_{1,2}^0 A_{1,2}^0 z_{1,1}^0 A_{1,1}^1 x_3^1 x_2^0 x_1^1 s_{1,3}^0 s_{1,2}^0 s_{1,1}^1 s_{2,3}^0 s_{2,2}^0 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1, r_{1,3}^0 r_{1,2}^0 r_{1,1}^0 r_{2,3}^0 r_{2,2}^1 r_{2,1}^0 r_{3,3}^0 r_{3,2}^0 r_{3,1}^0, A_{0,3}^0 A_{0,2}^0 A_{0,1}^0 z_{1,0}^0 z_{2,0}^0 z_{3,0}^0,$ $z_{3,3}^0 A_{3,3}^0 z_{3,2}^0 A_{3,2}^1 z_{3,1}^0 A_{3,1}^1 z_{2,3}^0 A_{2,3}^0 z_{2,2}^0 A_{2,2}^0 z_{2,1}^0 A_{2,1}^0 z_{1,3}^0 A_{1,3}^0 z_{1,2}^0 A_{1,2}^0 z_{1,1}^0 A_{1,1}^0 x_3^1 x_2^0 x_1^0 s_{1,3}^0 s_{1,2}^0 s_{1,1}^0 s_{2,3}^0 s_{2,2}^0 s_{2,1}^0 s_{3,3}^0 s_{3,2}^1 s_{3,1}^1 r_{1,3}^0 r_{1,2}^0 r_{1,1}^1 r_{2,3}^0 r_{2,2}^1 r_{2,1}^0 r_{3,3}^0 r_{3,2}^0 r_{3,1}^0 A_{0,3}^0 A_{0,2}^0 A_{0,1}^0 z_{1,0}^0 z_{2,0}^0 z_{3,0}^0\}$ |

When the *second* execution of **ParallelAdder**$(T_0, \alpha, \beta, \gamma, q, n)$ is called from step (4) of **Algorithm 1** in Section 3.8, tube $T_0$ with the result in Table 4 is regarded as the input tube. From the second parameter to the fourth parameter are replaced by the actual arguments $B$, $r$ and $y$ respectively. The values for the fifth and sixth parameters, $q$ and $n$, are both three. Because the value of $n$ is equal to three, three bits $B_{0,3}^0$, $B_{0,2}^0$ and $B_{0,1}^0$ are appended onto the tail of each bit pattern in tube $T_0$ after the execution of step (1a). Since the values for $n$ and $q$ are both three, step

(2a) will be executed three times and step (2c) will be executed nine times. The bit, $y_{1,0}^{0}$, from the first execution of step (2a) is appended onto the tail of each bit pattern in tube $T_0$. Next, after the first execution, the second execution and the third execution for step (2c), that call the algorithm, **ParallelOneBitAdder**($T_0$, $\alpha_{m-1,\,k}$, $\beta_{m,\,k}$, $\gamma_{m,\,k-1}$, $m$, $k$), are performed, tube $T_0$ = {$y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{1}\,z_{3,3}^{0}\,A_{3,3}^{1}\,z_{3,2}^{1}A_{3,2}^{1}\,z_{3,1}^{1}\,A_{3,1}^{0}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{1}\,z_{2,1}^{0}\,A_{2,1}^{1}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{1}\,x_3^{1}\,x_2^{1}\,x_1^{1}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{1}\,s_{2,3}^{0}\,s_{2,2}^{1}\,s_{2,1}^{0}\,s_{3,3}^{0}\,s_{3,2}^{1}\,s_{3,1}^{1}\,r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{0}\,r_{2,3}^{0}\,r_{2,2}^{0}\,r_{2,1}^{0}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{0}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}$, $y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{1}z_{3,3}^{0}\,A_{3,3}^{1}\,z_{3,2}^{1}A_{3,2}^{0}\,z_{3,1}^{1}\,A_{3,1}^{1}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{1}\,z_{2,1}^{0}\,A_{2,1}^{1}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{1}\,x_3^{0}\,x_2^{1}\,x_1^{0}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{1}\,s_{2,3}^{0}\,s_{2,2}^{1}\,s_{2,1}^{0}\,s_{3,3}^{0}\,s_{3,2}^{1}\,s_{3,1}^{1}$, $r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{1}\,r_{2,3}^{0}\,r_{2,2}^{0}\,r_{2,1}^{0}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{1}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}$, $y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{0}z_{3,3}^{0}\,A_{3,3}^{1}\,z_{3,2}^{1}A_{3,2}^{0}\,z_{3,1}^{1}\,A_{3,1}^{0}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{0}\,z_{2,1}^{0}\,A_{2,1}^{1}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{1}\,x_3^{1}\,x_2^{0}\,x_1^{1}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{1}\,s_{2,3}^{0}\,s_{2,2}^{0}\,s_{2,1}^{0}\,s_{3,3}^{0}\,s_{3,2}^{1}\,s_{3,1}^{1}$, $r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{0}\,r_{2,3}^{0}\,r_{2,2}^{1}\,r_{2,1}^{0}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{0}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}$, $y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{1}z_{3,3}^{0}\,A_{3,3}^{0}\,z_{3,2}^{0}A_{3,2}^{1}\,z_{3,1}^{0}\,A_{3,1}^{1}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{1}\,z_{2,1}^{0}\,A_{2,1}^{0}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{1}\,x_3^{0}\,x_2^{1}\,x_1^{0}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{1}\,s_{2,3}^{0}\,s_{2,2}^{0}\,s_{2,1}^{1}\,s_{3,3}^{0}\,s_{3,2}^{0}\,s_{3,1}^{1}$, $r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{0}\,r_{2,3}^{0}\,r_{2,2}^{0}\,r_{2,1}^{1}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{0}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}$, $y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{1}z_{3,3}^{0}\,A_{3,3}^{0}\,z_{3,2}^{0}A_{3,2}^{1}\,z_{3,1}^{0}\,A_{3,1}^{1}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{1}\,z_{2,1}^{0}\,A_{2,1}^{0}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{1}\,x_3^{0}\,x_2^{0}\,x_1^{1}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{1}\,s_{2,3}^{0}\,s_{2,2}^{0}\,s_{2,1}^{1}\,s_{3,3}^{0}\,s_{3,2}^{0}\,s_{3,1}^{1}$, $r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{0}\,r_{2,3}^{0}\,r_{2,2}^{0}\,r_{2,1}^{1}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{1}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}$, $y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{1}z_{3,3}^{0}\,A_{3,3}^{0}\,z_{3,2}^{1}A_{3,2}^{0}\,z_{3,1}^{0}\,A_{3,1}^{1}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{0}\,z_{2,1}^{1}\,A_{2,1}^{0}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{1}\,x_3^{0}\,x_2^{0}\,x_1^{1}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{1}\,s_{2,3}^{0}\,s_{2,2}^{0}\,s_{2,1}^{1}\,s_{3,3}^{0}\,s_{3,2}^{0}\,s_{3,1}^{1}\,r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{0}\,r_{2,3}^{0}\,r_{2,2}^{0}\,r_{2,1}^{0}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{0}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}$}. Similarly, after each operation in step (2) is performed, the result for tube $T_0$ is shown in Table 5.

**Lemma 4:** The **ParallelAdder**($T_0$ , $\alpha$, $\beta$, $\gamma$, $q$, $n$) procedure can be applied to perform a binary parallel adder of $n$ bits.

**Proof:** Refer to **Lemma 1**.

Table 5. The second execution result generated by **ParallelAdder**($T_0$, $\alpha$, $\beta$, $\gamma$, $q$, $n$).

| Tube | The second execution result generated by **ParallelAdder**($T_0$, $\alpha$, $\beta$, $\gamma$, $q$, $n$) |
|---|---|
| $T_0$ | {$y_{3,3}^{0}\,B_{3,3}^{0}\,y_{3,2}^{0}B_{3,2}^{0}\,y_{3,1}^{0}\,B_{3,1}^{0}\,y_{2,3}^{0}\,B_{2,3}^{0}\,y_{2,2}^{0}B_{2,2}^{0}\,y_{2,1}^{0}\,B_{2,1}^{0}\,y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{1}\,z_{3,3}^{0}\,A_{3,3}^{1}\,z_{3,2}^{1}A_{3,2}^{1}\,z_{3,1}^{1}\,A_{3,1}^{0}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{1}\,z_{2,1}^{0}\,A_{2,1}^{1}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{1}\,x_3^{1}\,x_2^{1}\,x_1^{1}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{1}\,s_{2,3}^{0}\,s_{2,2}^{1}\,s_{2,1}^{0}\,s_{3,3}^{0}\,s_{3,2}^{1}\,s_{3,1}^{1}\,r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{0}\,r_{2,3}^{0}\,r_{2,2}^{0}\,r_{2,1}^{0}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{0}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}\,y_{2,0}^{0}\,y_{3,0}^{0}$,<br>$y_{3,3}^{0}\,B_{3,3}^{0}\,y_{3,2}^{0}B_{3,2}^{0}\,y_{3,1}^{0}\,B_{3,1}^{1}\,y_{2,3}^{0}\,B_{2,3}^{0}\,y_{2,2}^{0}B_{2,2}^{0}\,y_{2,1}^{0}\,B_{2,1}^{1}\,y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{1}z_{3,3}^{0}\,A_{3,3}^{1}\,z_{3,2}^{1}A_{3,2}^{0}\,z_{3,1}^{1}\,A_{3,1}^{0}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{1}\,z_{2,1}^{0}\,A_{2,1}^{0}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{0}\,x_3^{1}\,x_2^{1}\,x_1^{0}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{0}\,s_{2,3}^{0}\,s_{2,2}^{1}\,s_{2,1}^{0}\,s_{3,3}^{0}\,s_{3,2}^{1}\,s_{3,1}^{1}$, $r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{1}\,r_{2,3}^{0}\,r_{2,2}^{0}\,r_{2,1}^{0}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{0}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}\,y_{2,0}^{0}\,y_{3,0}^{0}$,<br>$y_{3,3}^{0}\,B_{3,3}^{0}\,y_{3,2}^{0}B_{3,2}^{1}\,y_{3,1}^{0}\,B_{3,1}^{0}\,y_{2,3}^{0}\,B_{2,3}^{0}\,y_{2,2}^{0}B_{2,2}^{1}\,y_{2,1}^{0}\,B_{2,1}^{0}\,y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{0}z_{3,3}^{0}\,A_{3,3}^{1}\,z_{3,2}^{1}A_{3,2}^{0}\,z_{3,1}^{1}\,A_{3,1}^{0}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{0}\,z_{2,1}^{0}\,A_{2,1}^{1}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{1}\,x_3^{1}\,x_2^{0}\,x_1^{1}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{0}\,s_{2,3}^{0}\,s_{2,2}^{0}\,s_{2,1}^{0}\,s_{3,3}^{0}\,s_{3,2}^{1}\,s_{3,1}^{1}$, $r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{0}\,r_{2,3}^{0}\,r_{2,2}^{1}\,r_{2,1}^{0}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{0}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}\,y_{2,0}^{0}\,y_{3,0}^{0}$,<br>$y_{3,3}^{0}\,B_{3,3}^{0}\,y_{3,2}^{0}B_{3,2}^{1}\,y_{3,1}^{0}\,B_{3,1}^{1}\,y_{2,3}^{0}\,B_{2,3}^{0}\,y_{2,2}^{0}B_{2,2}^{0}\,y_{2,1}^{0}\,B_{2,1}^{0}\,y_{1,3}^{0}\,B_{1,3}^{0}\,y_{1,2}^{0}B_{1,2}^{0}\,y_{1,1}^{0}\,B_{1,1}^{1}z_{3,3}^{0}\,A_{3,3}^{0}\,z_{3,2}^{0}A_{3,2}^{1}\,z_{3,1}^{0}\,A_{3,1}^{1}\,z_{2,3}^{0}\,A_{2,3}^{0}\,z_{2,2}^{0}A_{2,2}^{0}\,z_{2,1}^{0}\,A_{2,1}^{0}\,z_{1,3}^{0}\,A_{1,3}^{0}\,z_{1,2}^{0}A_{1,2}^{0}\,z_{1,1}^{0}\,A_{1,1}^{1}\,x_3^{0}\,x_2^{0}\,x_1^{1}\,s_{1,3}^{0}\,s_{1,2}^{0}\,s_{1,1}^{1}\,s_{2,3}^{0}\,s_{2,2}^{0}\,s_{2,1}^{1}\,s_{3,3}^{0}\,s_{3,2}^{1}\,s_{3,1}^{1}\,r_{1,3}^{0}\,r_{1,2}^{0}\,r_{1,1}^{1}\,r_{2,3}^{0}\,r_{2,2}^{1}\,r_{2,1}^{0}\,r_{3,3}^{0}\,r_{3,2}^{0}\,r_{3,1}^{1}\,A_{0,3}^{0}\,A_{0,2}^{0}\,A_{0,1}^{0}\,z_{1,0}^{0}\,z_{2,0}^{0}\,z_{3,0}^{0}\,B_{0,3}^{0}\,B_{0,2}^{0}\,B_{0,1}^{0}\,y_{1,0}^{0}\,y_{2,0}^{0}\,y_{3,0}^{0}$} |

## 3.6. Constructing the Parallel One-bit XOR Operation on Bio-molecular Computing

The Exclusive-OR (XOR) operation of a bit for Boolean variables $A$ and $B$ generates an output of 1 if both $A$ and $B$ have different values and 0 if they are equal. The $\oplus$ symbol represents the XOR operation. The four possible combinations for the XOR operation are $0 \oplus 0 = 0$, $0 \oplus 1 = 1$, $1 \oplus 0 = 1$, $1 \oplus 1 = 0$. A truth table is usually used with logic operation to represent all possible

combinations of inputs and the corresponding outputs. The truth table for the XOR operation is shown in Table 6.

Table 6. Truth table for the XOR operation of a bit.

| Input | | output |
|---|---|---|
| A | B | C = A ⊕ B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Assume that two one-bit binary numbers, $A_{q,k}$ and $B_{q,k}$ for $1 \le k \le n$, are used to respectively represent the first input and the second input for the XOR operation of a bit. Also assume that $C_{q,k}$ for $1 \le k \le n$, is used to represent the output for the XOR operation of a bit. For the sake of convenience, assume that $A_{q,k}^{1}$ denotes that the value of $A_{q,k}$ is 1 and $A_{q,k}^{0}$ denotes that the value of $A_{q,k}$ is 0. Similarly, suppose that $B_{q,k}^{1}$ denotes that the value of $B_{q,k}$ is 1 and $B_{q,k}^{0}$ denotes that the value of $B_{q,k}$ is 0. Assume that $C_{q,k}^{1}$ denotes that the value of $C_{q,k}$ is 1 and $C_{q,k}^{0}$ denotes that the value of $C_{q,k}$ is 0. The following algorithm is used to perform the parallel one-bit XOR operation.

**Procedure ParalleOneBitXOR**($T_0$, $A_{q,k}$, $B_{q,k}$, $q$, $k$)

(1) $T_1 = +(T_0, A_{q,k}^{1})$ and $T_2 = -(T_0, A_{q,k}^{1})$
(2) $T_3 = +(T_1, B_{q,k}^{1})$ and $T_4 = -(T_1, B_{q,k}^{1})$
(3) $T_5 = +(T_2, B_{q,k}^{1})$ and $T_6 = -(T_2, B_{q,k}^{1})$
(4) **If** (Detect($T_3$) = = "yes") **then**
  (4a) Append-head($T_3$, $C_{q,k}^{0}$)
  **EndIf**
(5) **If** (Detect($T_4$) = = "yes") **then**
  (5a) Append-head ($T_4$, $C_{q,k}^{1}$)
  **EndIf**
(6) **If** (Detect($T_5$) = = "yes") **then**
  (6a) Append-head($T_5$, $C_{q,k}^{1}$)
  **EndIf**
(7) **If** (Detect($T_6$) = = "yes") **then**
  (7a) Append-head($T_6$, $C_{q,k}^{0}$)
  **EndIf**
(8) $T_0 = \cup(T_3, T_4, T_5, T_6)$
**EndProcedure**

**Lemma 5**: The **ParallelOneBitXOR**($T_0$, $A_{q,k}$, $B_{q,k}$, $q$, $k$) procedure can be used to implement the parallel XOR operation of one-bit.

**Proof**: The algorithm, **ParallelOneBitXOR**($T_0$, $A_{q,k}$, $B_{q,k}$, $q$, $k$), is implemented by the *extract*, *detect*, *merge* and *append-head* operations. Steps from (1) to (3) employ the *extract* operations to yield different tubes consisting of different inputs ($T_1$ to $T_6$). This implies, $T_1$ includes all of the inputs that have $A_{q,k} = 1$, $T_2$ contains all of the inputs that have $A_{q,k} = 0$, $T_3$ includes those inputs that have $A_{q,k} = 1$ and $B_{q,k} = 1$, $T_4$ consists of those inputs that have $A_{q,k} = 1$ and $B_{q,k} = 0$, $T_5$ comprises of those inputs that have $A_{q,k} = 0$ and $B_{q,k} = 1$, and $T_6$ includes those that have $A_{q,k} = 0$ and $B_{q,k} = 0$. After having performed *separation* operation from step (1) to (3), the results of XOR operation shown in Table 6 are poured into tubes $T_3$ through $T_6$ respectively.

Step (4a) through (7a) are applied to check whether it contains any input for tubes $T_3$, $T_4$, $T_5$, and $T_6$ or not respectively. If any a "yes" returned for those steps, then *append-head* operations will be performed correspondingly. Because tubes $T_3$, $T_4$, $T_5$ and $T_6$, subsequently, contains the input for the fourth row, the third row, the second row and the first row in Table 6. $C_{q,k}{}^0$ is appended onto the head of every input in tubes $T_3$ and $T_6$ if the detection result from step (4) or step (7) is yes. $C_{q,k}{}^1$ is subsequently appended onto the head of every input in tubes $T_4$ and $T_5$ when the detection result from step (5) or step (6) is yes. After performing step (4) through (7), we discover that one of four different outputs for the XOR operation of a bit as shown in Table 6 is correspondingly appended into tubes $T_3$ through $T_6$. The last execution of step (8) uses the *merge* operation to pour tubes $T_3$ through $T_6$ into tube $T_0$. Tube $T_0$ contains the result finishing the XOR operation of a bit as shown in Table 6.

## 3.7. Constructing the Parallel N-bit XOR Operation on Bio-molecular Computing

Simultaneously, the parallel XOR operation of $n$ bits generates the corresponding $n$-bit outputs for XOR operation with two $n$-bit Boolean variables $A$, represented by $A_{q,n}A_{q,n-1}\ldots A_{q,2}A_{q,1}$, and $B$, represented by $B_{q,n}B_{q,n-1}\ldots B_{q,2}B_{q,1}$. The following algorithm is proposed to perform the parallel $n$-bit XOR operation for $2^{q-1}$ partitions of a $q$-element set $S$.

**Procedure ParallelXOR**($T_0$, $A$, $B$, $q$, $n$)
(1) **For** $k$ = 1 **to** $n$
  (1a) **ParallelOneBitXOR**($T_0$, $A_{q,k}$, $B_{q,k}$, $q$, $k$).
**EndFor**
**EndProcedure**

When the algorithm, **ParallelXOR**($T_0$, $A$, $B$, $q$, $n$), is invoked from step (5) in **Algorithm 1**, tube $T_0$ with the result in Table 5 is regarded as the input tube. The second parameter is replaced by the total sum, $A$, for the value of each element in each $T$ and the third parameter is replaced by the total sum, $B$, for the value of each element in each $\bar{T}$. The fourth parameter $q$ and the fifth parameter $n$ are both three. Because the value of $n$ is equal to three, step (1a) will be executed three times. After the first execution of step (1a), that calls the algorithm, **ParallelOneBitXOR**($T_0$, $A_{q,k}$, $B_{q,k}$, $q$, $k$), is finished, tube $T_0 = \{C_{3,1}{}^0 \ldots B_{3,1}{}^0 \ldots A_{3,1}{}^0 \ldots x_3{}^1 x_2{}^1 x_1{}^1 \ldots, C_{3,1}{}^0 \ldots B_{3,1}{}^1 \ldots A_{3,1}{}^1 \ldots x_3{}^1 x_2{}^1 x_1{}^0 \ldots, C_{3,1}{}^0 \ldots B_{3,1}{}^0 \ldots A_{3,1}{}^0 \ldots x_3{}^1 x_2{}^0 x_1{}^1 \ldots, C_{3,1}{}^0 \ldots B_{3,1}{}^1 \ldots A_{3,1}{}^1 \ldots x_3{}^1 x_2{}^0 x_1{}^0 \ldots\}$. Then, after performing each execution of step (1a), the result for tube $T_0$ is shown in Table 7. Lemma 6 is applied to prove the correctness of the algorithm, **ParallelXOR**($T_0$, $A$, $B$, $q$, $n$).

**Lemma 6**: The procedure, **ParallelXOR**($T_0$, $A$, $B$, $q$, $n$), can be used to finish the parallel XOR operation of $n$ bits.

**Proof**: Tube $T_0$ is generated from the algorithm, **ParallelAdder**($T_0$, $\alpha$, $\beta$, $\gamma$, $q$, $n$) and contains those DNA strands representing the individual sum of two disjoint subsets, $T$ and $\bar{T}$ in each pair of ($T$ and $\bar{T}$) in $2^{q-1}$ partitions of a $q$-element set $S$. Step (1) is the single loop and is mainly applied to perform the function of parallel XOR operation of $n$ bits for $2^{q-1}$ partitions of a $q$-element set $S$. Each execution of step (1a) calls **ParallelOneBitXOR**($T_0$, $A_{q,k}$, $B_{q,k}$, $q$, $k$) to finish the XOR operation for the $k$th bit of two operands, $A$ and $B$, in tube $T_0$. After repeating the execution of step (1a) until the most significant bit of each operand, $A$ and $B$, is processed, tube $T_0$ includes the result of performing the parallel XOR operation of $n$ bits. In other words, after finishing the execution of single loop, tube $T_0$ contains the strands representing the results of parallel XOR operations for $2^{q-1}$ partitions of a $q$-element set $S$.

Table 7**.** The result generated by **ParallelXOR**($T_0$, $A$, $B$, $q$, $n$).

| Tube | The result generated by **ParallelXOR**($T_0$, $A$, $B$, $q$, $n$) |
|---|---|
| $T_0$ | $\{C_{3,3}^{\ 1}\ C_{3,2}^{\ 1}C_{3,1}^{\ 0}\ y_{3,3}^{\ 0}\ B_{3,3}^{\ 0}\ y_{3,2}^{\ 0}B_{3,2}^{\ 0}\ y_{3,1}^{\ 0}\ B_{3,1}^{\ 0}\ y_{2,3}^{\ 0}\ B_{2,3}^{\ 0}\ y_{2,2}^{\ 0}B_{2,2}^{\ 0}\ y_{2,1}^{\ 0}\ B_{2,1}^{\ 0}\ y_{1,3}^{\ 0}\ B_{1,3}^{\ 0}\ y_{1,2}^{\ 0}B_{1,2}^{\ 0}\ y_{1,1}^{\ 0}\ B_{1,1}^{\ 1}\ z_{3,3}^{\ 0}\ A_{3,3}^{\ 1}\ z_{3,2}^{\ 1}A_{3,2}^{\ 1}\ z_{3,1}^{\ 1}\ A_{3,1}^{\ 0}\ z_{2,3}^{\ 0}\ A_{2,3}^{\ 0}\ z_{2,2}^{\ 0}A_{2,2}^{\ 1}\ z_{2,1}^{\ 0}\ A_{2,1}^{\ 1}\ z_{1,3}^{\ 0}\ A_{1,3}^{\ 0}\ z_{1,2}^{\ 0}A_{1,2}^{\ 0}\ z_{1,1}^{\ 0}\ A_{1,1}^{\ 1}\ x_3^{\ 1}\ x_2^{\ 1}\ x_1^{\ 1}\ s_{1,3}^{\ 0}\ s_{1,2}^{\ 0}\ s_{1,1}^{\ 1}\ s_{2,3}^{\ 0}\ s_{2,2}^{\ 1}\ s_{2,1}^{\ 0}\ s_{3,3}^{\ 0}\ s_{3,2}^{\ 1}\ s_{3,1}^{\ 1}\ r_{1,3}^{\ 0}\ r_{1,2}^{\ 0}\ r_{1,1}^{\ 0}\ r_{2,3}^{\ 0}\ r_{2,2}^{\ 0}\ r_{2,1}^{\ 0}\ r_{3,3}^{\ 0}\ r_{3,2}^{\ 0}\ r_{3,1}^{\ 0}\ A_{0,3}^{\ 0}\ A_{0,2}^{\ 0}\ A_{0,1}^{\ 0}\ z_{1,0}^{\ 0}\ z_{2,0}^{\ 0}\ z_{3,0}^{\ 0}\ B_{0,3}^{\ 0}\ B_{0,2}^{\ 0}\ B_{0,1}^{\ 0}\ y_{1,0}^{\ 0}\ y_{2,0}^{\ 0}\ y_{3,0}^{\ 0}$, $C_{3,3}^{\ 1}\ C_{3,2}^{\ 0}C_{3,1}^{\ 0}\ y_{3,3}^{\ 0}\ B_{3,3}^{\ 0}\ y_{3,2}^{\ 0}B_{3,2}^{\ 0}\ y_{3,1}^{\ 0}\ B_{3,1}^{\ 1}\ y_{2,3}^{\ 0}\ B_{2,3}^{\ 0}\ y_{2,2}^{\ 0}B_{2,2}^{\ 0}\ y_{2,1}^{\ 0}\ B_{2,1}^{\ 1}\ y_{1,3}^{\ 0}\ B_{1,3}^{\ 0}\ y_{1,2}^{\ 0}B_{1,2}^{\ 0}\ y_{1,1}^{\ 0}\ B_{1,1}^{\ 1}\ z_{3,3}^{\ 0}\ A_{3,3}^{\ 1}\ z_{3,2}^{\ 1}A_{3,2}^{\ 0}\ z_{3,1}^{\ 1}\ A_{3,1}^{\ 1}\ z_{2,3}^{\ 0}\ A_{2,3}^{\ 0}\ z_{2,2}^{\ 0}A_{2,2}^{\ 1}\ z_{2,1}^{\ 0}\ A_{2,1}^{\ 0}\ z_{1,3}^{\ 0}\ A_{1,3}^{\ 0}\ z_{1,2}^{\ 0}A_{1,2}^{\ 0}\ z_{1,1}^{\ 0}\ A_{1,1}^{\ 0}\ x_3^{\ 1}\ x_2^{\ 1}\ x_1^{\ 0}\ s_{1,3}^{\ 0}\ s_{1,2}^{\ 0}\ s_{1,1}^{\ 0}\ s_{2,3}^{\ 0}\ s_{2,2}^{\ 1}\ s_{2,1}^{\ 0}\ s_{3,3}^{\ 0}\ s_{3,2}^{\ 1}\ s_{3,1}^{\ 1}, r_{1,3}^{\ 0}\ r_{1,2}^{\ 0}\ r_{1,1}^{\ 1}\ r_{2,3}^{\ 0}\ r_{2,2}^{\ 0}\ r_{2,1}^{\ 0}\ r_{3,3}^{\ 0}\ r_{3,2}^{\ 0}\ r_{3,1}^{\ 0}\ A_{0,3}^{\ 0}\ A_{0,2}^{\ 0}\ A_{0,1}^{\ 0}\ z_{1,0}^{\ 0}\ z_{2,0}^{\ 0}\ z_{3,0}^{\ 0}\ B_{0,3}^{\ 0}\ B_{0,2}^{\ 0}\ B_{0,1}^{\ 0}\ y_{1,0}^{\ 0}\ y_{2,0}^{\ 0}\ y_{3,0}^{\ 0}$, $C_{3,3}^{\ 1}\ C_{3,2}^{\ 1}C_{3,1}^{\ 0}\ y_{3,3}^{\ 0}\ B_{3,3}^{\ 0}\ y_{3,2}^{\ 0}B_{3,2}^{\ 1}\ y_{3,1}^{\ 0}\ B_{3,1}^{\ 0}\ y_{2,3}^{\ 0}\ B_{2,3}^{\ 0}\ y_{2,2}^{\ 0}B_{2,2}^{\ 1}\ y_{2,1}^{\ 0}\ B_{2,1}^{\ 0}\ y_{1,3}^{\ 0}\ B_{1,3}^{\ 0}\ y_{1,2}^{\ 0}B_{1,2}^{\ 0}\ y_{1,1}^{\ 0}\ B_{1,1}^{\ 0}z_{3,3}^{\ 0}\ A_{3,3}^{\ 1}\ z_{3,2}^{\ 1}A_{3,2}^{\ 0}\ z_{3,1}^{\ 1}\ A_{3,1}^{\ 0}\ z_{2,3}^{\ 0}\ A_{2,3}^{\ 0}\ z_{2,2}^{\ 0}A_{2,2}^{\ 0}\ z_{2,1}^{\ 0}\ A_{2,1}^{\ 1}\ z_{1,3}^{\ 0}\ A_{1,3}^{\ 0}\ z_{1,2}^{\ 0}A_{1,2}^{\ 0}\ z_{1,1}^{\ 0}\ A_{1,1}^{\ 1}\ x_3^{\ 1}\ x_2^{\ 0}\ x_1^{\ 1}\ s_{1,3}^{\ 0}\ s_{1,2}^{\ 0}\ s_{1,1}^{\ 1}\ s_{2,3}^{\ 0}\ s_{2,2}^{\ 0}\ s_{2,1}^{\ 0}\ s_{3,3}^{\ 0}\ s_{3,2}^{\ 1}\ s_{3,1}^{\ 1}, r_{1,3}^{\ 0}\ r_{1,2}^{\ 0}\ r_{1,1}^{\ 0}\ r_{2,3}^{\ 0}\ r_{2,2}^{\ 1}\ r_{2,1}^{\ 0}\ r_{3,3}^{\ 0}\ r_{3,2}^{\ 0}\ r_{3,1}^{\ 0}, A_{0,3}^{\ 0}\ A_{0,2}^{\ 0}\ A_{0,1}^{\ 0}\ z_{1,0}^{\ 0}\ z_{2,0}^{\ 0}\ z_{3,0}^{\ 0}\ B_{0,3}^{\ 0}\ B_{0,2}^{\ 0}\ B_{0,1}^{\ 0}\ y_{1,0}^{\ 0}\ y_{2,0}^{\ 0}\ y_{3,0}^{\ 0}$, $C_{3,3}^{\ 0}\ C_{3,2}^{\ 0}C_{3,1}^{\ 0}\ y_{3,3}^{\ 0}\ B_{3,3}^{\ 0}\ y_{3,2}^{\ 0}B_{3,2}^{\ 1}\ y_{3,1}^{\ 0}\ B_{3,1}^{\ 1}\ y_{2,3}^{\ 0}\ B_{2,3}^{\ 0}\ y_{2,2}^{\ 0}B_{2,2}^{\ 1}\ y_{2,1}^{\ 0}\ B_{2,1}^{\ 1}\ y_{1,3}^{\ 0}\ B_{1,3}^{\ 0}\ y_{1,2}^{\ 0}B_{1,2}^{\ 0}\ y_{1,1}^{\ 0}\ B_{1,1}^{\ 1}\ z_{3,3}^{\ 0}\ A_{3,3}^{\ 0}\ z_{3,2}^{\ 0}A_{3,2}^{\ 1}\ z_{3,1}^{\ 0}\ A_{3,1}^{\ 1}\ z_{2,3}^{\ 0}\ A_{2,3}^{\ 0}\ z_{2,2}^{\ 0}A_{2,2}^{\ 0}\ z_{2,1}^{\ 0}\ A_{2,1}^{\ 0}\ z_{1,3}^{\ 0}\ A_{1,3}^{\ 0}\ z_{1,2}^{\ 0}A_{1,2}^{\ 0}\ z_{1,1}^{\ 0}\ A_{1,1}^{\ 0}\ x_3^{\ 1}\ x_2^{\ 0}\ x_1^{\ 0}\ s_{1,3}^{\ 0}\ s_{1,2}^{\ 0}\ s_{1,1}^{\ 0}\ s_{2,3}^{\ 0}\ s_{2,2}^{\ 0}\ s_{2,1}^{\ 0}\ s_{3,3}^{\ 0}\ s_{3,2}^{\ 1}\ s_{3,1}^{\ 1}\ r_{1,3}^{\ 0}\ r_{1,2}^{\ 0}\ r_{1,1}^{\ 1}\ r_{2,3}^{\ 0}\ r_{2,2}^{\ 1}\ r_{2,1}^{\ 0}\ r_{3,3}^{\ 0}\ r_{3,2}^{\ 0}\ r_{3,1}^{\ 0}\ A_{0,3}^{\ 0}\ A_{0,2}^{\ 0}\ A_{0,1}^{\ 0}\ z_{1,0}^{\ 0}\ z_{2,0}^{\ 0}\ z_{3,0}^{\ 0}\ B_{0,3}^{\ 0}\ B_{0,2}^{\ 0}\ B_{0,1}^{\ 0}\ y_{1,0}^{\ 0}\ y_{2,0}^{\ 0}\ y_{3,0}^{\ 0}\}$ |

## 3.8. A DNA Algorithm for Solving the Set Partition Problem

The following algorithm is used to solve the set-partition problem of a $q$-element set $S$. Notations used in the following algorithm are denoted in the previous sections.

**Algorithm 1**: Solving the set-partition problem.

(1) **Init** ($T_0$, $q$)
(2) **Value** ($T_0$, $q$, $n$)
(3) **ParallelAdder**($T_0$, $A$, $s$, $z$, $q$, $n$)
(4) **ParallelAdder**($T_0$, $B$, $r$, $y$, $q$, $n$)
(5) **ParallelXOR**($T_0$, $A$, $B$, $q$, $n$)
(6) **For** $k$ = 1 **to** $n$
    (6a) $T_1$ = + ($T_0$, $C_{q,k}^{\ 0}$) and $T_2$= −($T_0$, $C_{q,k}^{\ 0}$)
    (6b) $T_0$ = ∪($T_1$, $T_0$)
    (6c) Discard ($T_2$)
  **EndFor**
(7) **If** (Dectect ($T_0$) = = ”yes”) **then**
    (7a) Read($T_0$)
  **EndIf**
**EndAlgorithm**

Consider that **Algorithm 1** is used to solve the set-partition problem of a set $S$ with {001, 010, 011}. When step (1) is performed, it calls the algorithm, **Init** ($T_0$, $q$). The first actual argument, tube $T_0$, is an empty tube and the second actual argument, $q$ is equal to three. After all operations in **Init** ($T_0$, $q$) is performed, the result for tube $T_0$ is shown in Table 1. Then, step (2) calls the

algorithm, **Value** ($T_0$, $q$, $n$). Tube $T_0$ with the result in Table 1 is regarded as the first actual argument, and the second actual argument and the third actual argument are equal to three. After each operation in **Value** ($T_0$, $q$, $n$) is performed, the result for tube $T_0$ is shown in Table 2.

Next, step (3) calls the algorithm, **ParallelAdder** ($T_0$, $A$, $s$, $z$, $q$, $n$). Tube $T_0$ with the result in Table 2 is regarded as the first actual argument. The second actual argument $A$ is used to represent the total sum for the value of each element in $T$, which is in the four partitions of a three-element set $S$. The third actual argument $s$ is used to represent the value of each element in $T$. The fourth actual argument $z$ is used to represent the carry for performing addition of each element in $T$. The fifth actual argument $q$ and the sixth actual argument $n$ are equal to three. After each operation in **ParallelAdder**($T_0$, $A$, $s$, $z$, $q$, $n$) is performed, the result for tube $T_0$ is shown in Table 4.

Then, step (4) calls the algorithm, **ParallelAdder**($T_0$, $B$, $r$, $y$, $q$, $n$). Tube $T_0$ with the result in Table 4 is regarded as the first actual argument. The second actual argument $B$ is used to represent the total sum for the value of each element in $\bar{T}$ , which is in the four partitions of a three-element set $S$. The third actual argument $r$ is used to represent the value of each element in $\bar{T}$ The fourth actual argument $y$ is used to represent the carry for performing addition of each element in $\bar{T}$ The fifth actual argument $q$ and the sixth actual argument $n$ are equal to three. After each operation in **ParallelAdder** ($T_0$, $B$, $r$, $y$, $q$, $n$) is finished, the result for tube $T_0$ is shown in Table 5.

Then, step (5) calls the algorithm, **ParallelXOR**($T_0$, $A$, $B$, $q$, $n$). Tube $T_0$ with the result in Table 5 is regarded as the first actual argument. The second actual argument $A$ is used to represent the total sum for the value of each element in $T$. The third actual argument $B$ is used to represent the total sum for the value of each element in $\bar{T}$ .The fourth actual argument $q$ and the fifth actual argument $n$ are equal to three. After each operation in **ParallelXOR**($T_0$, $A$, $B$, $q$, $n$) is finished, the result for tube $T_0$ is shown in Table 7.

step (6) is a single loop and is used to search the answer of the set-partition problem. Because $C_{3,1}^{\ 0}$ appears in each bit pattern in tube $T_0$ in Table 7, after the first execution of step (6a) is performed, tube $T_1$ obtains the same result in Table 7 and tube $T_0 = \varnothing$, tube $T_2 = \varnothing$. Then, after the first execution of step (6b) is finished, tube $T_1 = \varnothing$, and tube $T_0$ obtains the same result in Table 7. After the first execution of step (6c) is finished, tube $T_2 = \varnothing$. Similarly, after each operation in step (6) is finished, the result for tube $T_0$ is shown in Table 8. Finally, after the execution of step (7) is performed, the answer is that $T = \{3\}$ and $\bar{T} = \{1, 2\}$. Theorem 1 is used to prove the correctness of Algorithm 1.

Table 8. The final result generated by Algorithm 1.

| Tube | The final result generated by **Algorithm 1** |
|---|---|
| $T_0$ | $\{C_{3,3}^{\ 0}\ C_{3,2}^{\ 0}C_{3,1}^{\ 0}\ y_{3,3}^{\ 0}\ B_{3,3}^{\ 0}\ y_{3,2}^{\ 0}B_{3,2}^{\ 1}\ y_{3,1}^{\ 0}\ B_{3,1}^{\ 1}\ y_{2,3}^{\ 0}\ B_{2,3}^{\ 0}\ y_{2,2}^{\ 0}B_{2,2}^{\ 1}\ y_{2,1}^{\ 0}\ B_{2,1}^{\ 1}\ y_{1,3}^{\ 0}\ B_{1,3}^{\ 0}\ y_{1,2}^{\ 0}B_{1,2}^{\ 0}\ y_{1,1}^{\ 0}\ B_{1,1}^{\ 1}z_{3,3}^{\ 0}\ A_{3,3}^{\ 0}\ z_{3,2}^{\ 0}A_{3,2}^{\ 1}\ z_{3,1}^{\ 0}\ A_{3,1}^{\ 1}\ z_{2,3}^{\ 0}\ A_{2,3}^{\ 0}\ z_{2,2}^{\ 0}A_{2,2}^{\ 0}\ z_{2,1}^{\ 0}\ A_{2,1}^{\ 0}\ z_{1,3}^{\ 0}\ A_{1,3}^{\ 0}\ z_{1,2}^{\ 0}A_{1,2}^{\ 0}\ z_{1,1}^{\ 0}\ A_{1,1}^{\ 0}\ x_3^{\ 1}\ x_2^{\ 0}\ x_1^{\ 0}\ s_{1,3}^{\ 0}\ s_{1,2}^{\ 0}\ s_{1,1}^{\ 0}\ s_{2,3}^{\ 0}\ s_{2,2}^{\ 0}\ s_{2,1}^{\ 0}\ s_{3,3}^{\ 0}\ s_{3,2}^{\ 1}\ s_{3,1}^{\ 1}\ r_{1,3}^{\ 0}\ r_{1,2}^{\ 0}\ r_{1,1}^{\ 1}\ r_{2,3}^{\ 0}\ r_{2,2}^{\ 1}\ r_{2,1}^{\ 0}\ r_{3,3}^{\ 0}\ r_{3,2}^{\ 0}\ r_{3,1}^{\ 0}\ A_{0,3}^{\ 0}\ A_{0,2}^{\ 0}\ A_{0,1}^{\ 0}\ z_{1,0}^{\ 0}\ z_{2,0}^{\ 0}\ z_{3,0}^{\ 0}\ B_{0,3}^{\ 0}\ B_{0,2}^{\ 0}\ B_{0,1}^{\ 0}\ y_{1,0}^{\ 0}\ y_{2,0}^{\ 0}\ y_{3,0}^{\ 0}\}$ |

**Theorem 1**: From those steps in **Algorithm 1**, the set partition problem for $2^{q-1}$ partitions of a $q$-element set $S$ can be solved.

**Proof:** On the execution of step (1), it calls **Init** ($T_0$, $q$) to construct solution space for $2^{q-1}$

partitions of a $q$-element set $S$. This means that the tube $T_0$ includes strands encoding $2^{q-1}$ partitions. Next the execution of step (2) calls **Value** ($T_0$, $q$, $n$) to encode the value of each element in $2^{q-1}$ partitions. On the execution of step (3), it calls **ParallelAdder**($T_0$, $A$, $s$, $z$, $q$, $n$) to perform the function of a parallel adder for computing the sum to the value of each element in $T$ in each pair of ($T$ and $\bar{T}$ ). Similarly, on the execution of step (4), it calls **ParallelAdder**($T_0$, $B$, $r$, $y$, $q$, $n$) to perform the function of a parallel adder for computing the sum to the value of each element in $\bar{T}$ in each pair of ($T$ , $\bar{T}$ ). Then, after step (5) is executed, it calls **ParallelXOR**($T_0$, $A$, $B$, $q$, $n$) to perform the function of parallel XOR operation of $n$ bits for the total sum to each pair of ($T$ ,$\bar{T}$ ).

Then, step (6) is a single loop and is employed individually to retrieve those DNA strands encoded zero value as the result of parallel XOR operation of $n$ bits. Each time step (6a) uses the *extract* operation to form two test tubes: $T_1$ and $T_2$ .The first tube $T_1$ includes all of the strands that have $C_{q,k}$=0. The second tube $T_2$ contains all of the strands that have $C_{q,k}$=1. Then, on each execution of step (6b), it employs the *merge* operation to pour tube $T_1$ into $T_0$ and each execution of step (6c) uses the *discard* operation to discard tube $T_2$ containing the result of parallel XOR operation is *not* equal to zero. After each step in the single loop is performed $n$ times, tube $T_0$ includes DNA strands that encodes the sum of subset $T$ to be equal to that of its exclusive subset $\bar{T}$ , the answer for the set-partition is found from tube $T_0$. Finally, the execution of step (7) uses the *detect* operation to check if there is any DNA strand in tube $T_0$. If it returns a "yes", then the execution of step (7a) applies the *read* operation to read the answer. Otherwise, no solution exists. Therefore, the set-partition problem for $2^{q-1}$ partitions of a $q$-element set $S$ can be computed from those steps in Algorithm 1.

### 3.9 The Complexity of Algorithm 1

**Theorem 2:** Suppose that a finite set $S$ is $\{s_1, s_2, \ldots, s_q\}$. The set partition problem for $S$ can be solved with O ($q \times n$) biological operations, O ($2^n$) library strands, O (1) tubes and the longest library strands, O ($q \times n$), where $n$ is the number of bits for representing the value of each element in $S$.
**Proof:** refer to Algorithm 1.

## 4. Conclusions

In this paper, based on biological operations, we propose DNA-based algorithms for solving the set partition problem of a $q$-element set. The computational complexity is clearly reduced to O ($q$*$n$), where $q$ is the numbers of element in set $S$ we concerned, $n$ is the number of bits for representing the value of each element in $S$.

Although the future of molecular computers is vague at present, it is likely that in the future DNA computers will be the better choice for performing massively parallel computations. However, there are still severe challenges to face and many technical difficulties to overcome before this becomes a reality. We hope that this paper helps to manifest that molecular computing is a technology worth pursuing.

## Acknowledgements

The author is very grateful to Dr. Amos who is the author of the 5th reference and the 6th reference for proposing valuable information about implementation of biological operations.

# References


[1] Adleman, L. M. (1994) "Molecular computation of solutions to combinatorial problems", Science, 266, November 11, 1021-1024.

[2] Adleman, L.M., Rothemund, P. W. K., Roweis S., and Winfree, E. (1999) "On applying molecular computation to the Data Encryption Standard", The 2nd annual workshop on DNA Computing, Princeton University, DIMACS: series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 31-44.

[3] Adleman,L.M., Braich, R.S., Johnson, C., Rothemund, P.W.K., Hwang, D. and Chelyapov, N. (2002) "Solution of a 20-Variable 3-SAT Problem on a DNA Computer", Science, Volume 296, Issue 5567, 499-502, 19 April.

[4] Ahrabian, H. and Nowzari-Dalini,A. (2004) "DNA simulation of nand Boolean circuits", Advanced Modeling and Optimization, Volume 6, Number 2, 33-41.

[5] Amos, M. (1997) DNA Computation, Ph.D. Thesis. department of computer science, the University of Warwick.

[6] Amos,M.(2005) Theoretical and Experimental DNA Computation. Springer, ISBN3540657738.

[7] Blackburn, G. M. and Gait, M. J. (1990) Nucleic Acids in Chemistry and Biology. IRL Press.

[8] Boneh, D., Dunworth, C., Lipton, R. J. and Sgall J., 1996. "On the Computational Power of DNA", Discrete Applied Mathematics, Special Issue on Computational Molecular Biology, Volume 71, 79-94.

[9] Boneh, D., Dunworth, C., and Lipton, R. J. (1996) "Breaking DES using a molecular computer", In Proceedings of the 1st DIMACS Workshop on DNA Based Computers, 1995, American Mathematical Society. In DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 27, 37-66.

[10] Braich, R. S., Johnson, C., Rothemund, P.W.K., Hwang, D., Chelyapov, N. and Adleman, L. M. (2001) "Solution of a satisfiability problem on a gel-based DNA computer", Proceedings of the 6th International Conference on DNA Computation in the Springer-Verlag Lecture Notes in Computer Science series, 1-27.

[11] Chang, W.L., Ho, M. and Guo, M. (2004) "Molecular solutions for the subset-sum problem on DNA-based supercomputing", BioSystems, Volume 73, No. 2, 117-130.

[12] Chang, W.L., Ho, M. and Guo, M. (2005) "Fast parallel molecular algorithms for DNA-based computation: factoring integers", IEEE Transactions on Nanobioscience, Volume 4, No. 2, 149-163.

[13] Cormen, T. H., Leiserson, C. E. and Rivest, R. L. (2001) Introduction to algorithms (the second edition). MIT Press.

[14] Cukras, A. R., Faulhammer,D., Lipton, R.J. and Landweber, L. F. (1998) "Chess games: A model for RNA-based computation", In Proceedings of the 4th DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16-19, 27-37.

[15] Eckstein F. (1991) Oligonucleotides and Anologues. Oxford University Press.

[16] Feynman, R. P. (1961) "In miniaturization", D.H. Gilbert, Ed., Reinhold Publishing Corporation, New York, 282-296.

[17] Garey, M. R. and Johnson, D. S. (1979) Computer and intractability. Freeman, San Fransico, CA.

[18] Guarnieri,F., Fliss, M. and Bancroft, C. (1996) "Making DNA add", Science, Vol. 273, 220–223.

[19] Guo, M., Chang, W. L., Ho, M., Lu, J. and Cao, J. (2005) "Is optimal solution of every NP-complete or NP-hard problem determined from its characteristic for DNA-based computing", Biosystems, Vol. 80, No. 1. 71-82.

[20] Ho, Michael (2005) "Fast parallel molecular solutions for DNA-based supercomputing: the subset-product problem", BioSystems, Volume 80, 233–250.

[21] Lipton, R. J. (1995) "DNA solution of hard computational problems", Science, 268, 542:545.

[22] Paun,G., Rozenberg, G. and Salomaa, A. (1998) DNA Computing: New Computing Paradigms. Springer-Verlag, New York.

[23] Quyang, Q. P., Kaplan, Liu, D. S. and Libchaber A. (1997) "DNA solution of the maximal clique problem", Science, 278:446-449.

[24] Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N. V., Goodman, M. F., Rothemund, Paul W.K. and Adleman, L. M. (1999) "A sticker based model for DNA computation", 1999, 2nd annual workshop on DNA Computing, Princeton University. Eds. L. Landweber and E. Baum, DIMACS:

series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1-29.
[25] Schuster, A. (2005) DNA databases. BioSystems, Volume 81, 234–246.
[26] Sinden, R. R. (1994) DNA Structure and Function. Academic Press.
[27] Watson,J., Gilman, M., Witkowski,J. and Zoller, M. (1992) Recombinant DNA (2nd edition). Scientific American Books, W. H. Freeman and Co..
[28] Watson, J., Hoplins, N., Roberts, J. and et al. (1987) Molecular Biology of the Gene. Benjamin/Cummings Menlo Park CA.
[29] Xiao, D., Li, W., Zhang, Z. and He, L. (2005) "Solving the maximum cut problems in the Adleman–Lipton model", BioSystems, Volume 82, 203-207.
[30] Yeh, C.W., Chu, C. P. and Wu, K.R. (2006) "Molecular solutions to the binary integer programming problem based on DNA computation", Biosystems, Volume: 83, Issue: 1, January, 56-66.

## Authors

**Sientang Tsai** received his B.S. from department of physics, National Taiwan Normal University in 1974 and got M.S. degree in computer science from University of Georgia, Athens, U.S.A. in 1983. He is currently an associate professor at the department of information management in Southern Taiwan University of Science and Technology. His researching interests include parallel computing, quantum computing and DNA computing.

**Wei-Yeh Chen** received his B.S. degree from National Cheng Kung University in 1986, M.S. degree from National Chiao Tung University in 1988, and Ph. D. degree in information management from National Taiwan University of Science and Technology in 2004. From 1990 to 2004, he was a lecturer in the Department of Electronic Engineering at Northern Taiwan Institute of Science and Technology. From 2004 to 2005, he was an associate professor of the Institute. Since August 2005, he has been an associate professor in the Department of Information Management, Southern Taiwan University of Science and Technology, Tainan, Taiwan. His current research interests are resource allocation of mobile communications systems, protocol design of wireless mesh networks, and spectrum management of cognitive radio networks.