Proc. of the Intl. Conf. on Advances In Bio-Informatics And Environmental Engineering - ICABEE 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-019-4 doi: 10.15224/ 978-1-63248-019-4-03

A Fast Bioinformatics Approach for Solving Backtracking of DNA Sequence Evolution in One Dimensional Cellular Automata

Michael Shan-Hui Ho, Paul Pin-Shuo Huang, Kun-Yu Hung, Kevin Kai-Wen Cheng, and Elizabeth Hsin-Yu Li

***Abstract*—It is a well-known fact that the DNA mutation plays a very important role in DNA sequence evolution. The backtracking problem of DNA sequence evolution in one dimensional cellular automata (CA) has ben recognized as a NP problem. In this research, a newly developed bioinformatics approach constructs a DNA sequence evolution model in using one dimensional cellular automata. Its corresponding backtracking of DNA sequence evolution is accomplished by an order-finding bioinformatics algorithm for efficient operations. The time complexity of a proposed bioinformatics approach for DNA sequence evolution in one dimensional cellular automata is found in O(n^2) polynomial bound. Our newly developed algorithms for solving backtracking of DNA sequence evolution in one dimensional CA are also in O(n^2) polynomial bound.**

***Keywords*: DNA sequence evolution, DNA mutation, Cellular Automata, Bioinformatics,Order-finding.**

1 Introduction

DNA is the basic and necessary unit for any creature on earth which records all of information about its characteristics, lifestyle and genetic information. It lets all creatures adapt various environments by evolution.

It is a well-known fact that the DNA mutation plays a very important role in DNA sequence evolution. In this paper, we construct a bioinformatics approach for DNA sequence evolution in one dimensional cellular automata (CA) and solve backtracking of DNA sequence evolution. It's important to understand the origin or evolution steps of DNA sequences which can help us to find or to backtrack their characteristics or differences. By recognizing these features, we can clearly find some DNA changes or mutations in creatures, normal cells, or damaged cells.

2 DNA Sequence Evolution

Knowledge of DNA sequences has become indispensable for basic biological research, such as diagnostic, biotechnology, forensic biology and biological systematic.

In DNA evolution, we define an evolution event is the same as a change in state, which may occur in one or more cellular automaton (CA) cells. Therefore, mutation is an evolution event and it corresponds to cell state changes. The time step in the CA evolution is the time interval between two CA cell changes and, therefore, the time flow is not uniform.

In this apporach, it is supposed that the state of each cell has changed as a result of the effect of the states of its neighbors. The new state of the ith cell at next time step (generally the s+1 step) can be shown in Equation (1) [1]:

$$C_i^{s+1} = \widehat{W}\left(C_{i-j}^s, \dots, C_{i-3}^s, C_{i-2}^s, C_{i-1}^s, C_i^s, C_{i+1}^s, C_{i+2}^s C_{i+3}^s, \dots, C_{i+j}^s\right) \quad (1)$$

In Equation (1), matrix ,$\widehat{W}$, is a linear evolution rule matrix shown in the following:

$$\begin{bmatrix} \vdots \\ C_{i-2}^{t+1} \\ C_{i-1}^{t+1} \\ C_i^{t+1} \\ C_{i+1}^{t+1} \\ C_{i+2}^{t+1} \\ \vdots \end{bmatrix} = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & W_{i-2,j-2} & W_{i-2,j-1} & W_{i-2,j} & W_{i-2,j+1} & W_{i-2,j+2} & \cdots \\ \cdots & W_{i-1,j-2} & W_{i-1,j-1} & W_{i-1,j} & W_{i-1,j+1} & W_{i-1,j+2} & \cdots \\ \cdots & W_{i,j-2} & W_{i,j-1} & W_{i,j} & W_{i,j+1} & W_{i,j+2} & \cdots \\ \cdots & W_{i+1,j-2} & W_{i+1,j-1} & W_{i+1,j} & W_{i+1,j+1} & W_{i+1,j+2} & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ C_{i-2}^{t} \\ C_{i-1}^{t} \\ C_i^{t} \\ C_{i+1}^{t} \\ C_{i+2}^{t} \\ \vdots \end{bmatrix} \quad (2)$$

In equation (1) cell states are one of the four bases A, C, T and G, which are represented by numbers of the quaternary number system, which contains only four numbers, i.e. 0, 1, 2 and 3. We represent the bases using the follwoing numbers: A as 0, C as 1, T as 2, and G as 3.

For example, a very small DNA strand which at time t has seven bases: { A, A, C, T, A, G, T }. This strand is represented by the following numbers: {0, 0, 1, 2, 0, 3, 2}. Suppose that this DNA strand evolves according to the following evolution rule matrix: $\widehat{W}=\begin{bmatrix} 1000000 \\ 0100000 \\ 0010000 \\ 0011100 \\ 0001100 \\ 0000010 \\ 0000001 \end{bmatrix}$, C=$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \\ 0 \\ 3 \\ 2 \end{bmatrix}$

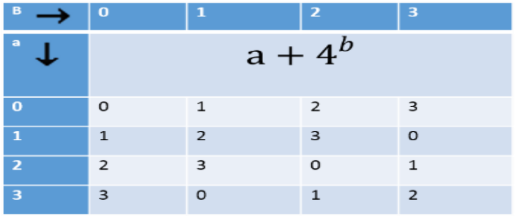
In Equation (2), the CA state at the next time step is calculated as follows:

$$C^{S+1} = \widehat{W} \times C^S$$

It is reminded that the additions are modulo 4.

The Caley's Table using modulo 4 additions is shown in Table 1.

Table 1: Caley's Table using the modulo 4 additions



B →	0	1	2	3
a ↓	$a + 4^b$			
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

3 Backtracking of DNA Sequence Evolution

If each cell has only on and off states, one dimensional CA is simply from a initial configuration of width n cells evolved to 2^n different configurations [2]. In this paper, given a specific initial configuration of width n cells in one dimensional cellular automaton, each DNA sequence

Michael Shan-Hui Ho
Department of Electrical Engineering, NTPU,
New Taipei City, Taiwan, ROC
Paul Pin-Shuo Huang
Department of Electrical Engineering, NTPU,
New Taipei City, Taiwan, ROC
Kun-Yu Hung
Department of Information Management, MCU
Taoyuan Country, Taiwan, ROC
Kevin Kai-Wen Cheng
Department of Electrical Engineering, NTPU,
New Taipei City, Taiwan, ROC
Elizabeth Hsin-Yu Li
Department of Electrical Engineering, NTPU,
New Taipei City, Taiwan, ROC

SEEK DIGITAL LIBRARY

evolution can have 4^n different configurations which are responsible for processing all the computational basis states. Next, it is demonstrated that using a modified Shor's order-finding algorithm [3] to solve the backtracking problem of DNA sequence evolution in one dimensional cellular automata is implemented. Then, it is also proved that a measurement on the answer for solving backtracking of DNA sequence is the same as that of Shor's order-finding algorithm or similar to that of the breaktrough of a RSA cryptosystem [4].

4 DNA Manipulations

DNA Manipulations are used in Adleman-Lipton model shown inn this subsection. The DNA Model of computation has eight biological operations shown in the following:

1. ***Extract.*** Given a tube *P* and a short single strand of DNA, *S*, the operation produces two tubes +(*P*, *S*) and –(*P*, *S*), where +(*P*, *S*) is all of the molecules of DNA in *P* which contain *S* as a sub-strand and–(*P*, *S*) is all of the molecules of DNA in *P* which do not contain *S*.
2. ***Merge***. Given tubes P_1 and P_2, yield $\cup(P_1, P_2)$, where $\cup(P_1, P_2) = P_1 \cup P_2$. This operation is used to pour two tubes into one, without any change in the individual strands.
3. ***Detect***. Given a tube *P*, if *P* includes at least one DNA molecule we have 'yes', and if *P* contains no DNA molecule we have 'no'.
4. ***Discard***. Given a tube *P*, the operation discards *P*.
5. ***Amplify***. Given a tube *P*, the operation, *Amplify* (P, P_1, P_2), will produce two new tubes P_1 and P_2 so that P_1, and P_2 are totally a copy of *P* (P_1, and P_2 are now identical) and *P* becomes an empty tube.
6. ***Append***. Given a tube *P* containing a short strand of DNA, *Z*, and the operation will append *A* onto the end of every strand in *P*.
7. ***Append-head.*** Given a tube *P* containing a short strand of DNA, *Z*, and the operation will append *A* onto the head of every strand in *P*.
8. ***Read***. Given a tube P, the operation is used to describe a single molecule, which is contained in tube P.

5 Basic Bioinformatics Circuitry

We use logic truth tables to optimize and complete logic bio-circuit operations that can construct most basic DNA logic circuits. These DNA logic circuits (gates) gates are AND, OR, XOR, etc.

5.1 AND Operation on Bioinformatics Computing

The AND operation of a bit with two input Boolean variables U and V generates a result of 1 or 0. The logic circuitry of parallel AND on one-bit is shown in Figure 1. The corresponding truth table of the one-bit AND is shown in Table 2.

Table 2: The truth table of the one-bit AND

Input		**Output**
U_k	V_k	$AND_k = U_k \wedge V_k$
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**

Figure 1: Logic circuitry of parallel AND on one-bit

```
**ParallelOneBitAND**($T_0$, $U_k$, $V_k$, $AND_k$ )
$T_1^{U=1} = +(T_0, U_k^1)$ and $T_1^{U=0} = -(T_0, U_k^1)$.
$T_2^{U=1,V=1} = +(T_1^{U=1}, V_k^1)$ and $T_2^{U=1,V=0} = -(T_1^{U=1}, V_k^1)$
$T_2^{U=0,V=1} = +(T_1^{U=0}, V_k^1)$ and $T_2^{U=0,V=0} = -(T_1^{U=0}, V_k^1)$
If (Detect($T_2^{U=1,V=1}$) = = "yes") then
  Append-head($T_2^{U=1,V=1}$, $AND_k^1$)  EndIf
If (Detect($T_2^{U=1,V=0}$) = = "yes") then
  Append-head($T_2^{U=1,V=0}$, $AND_k^0$)  EndIf
If (Detect($T_2^{U=0,V=1}$) = = "yes") then
    Append-head($T_2^{U=0,V=1}$, $AND_k^0$)  EndIf
If (Detect($T_2^{U=0,V=0}$) = = "yes") then
    Append-head($T_2^{U=0,V=0}$, $AND_k^0$)  EndIf
$T_0 = \cup(T_2^{U=1,V=1}, T_2^{U=1,V=0}, T_2^{U=0,V=1}, T_2^{U=0,V=0})$
EndAlgorithm
```

Figure 2: Parallel AND operation on one bit algorithm

5.2 OR Operation on Bioinformatics Computing

The OR operation of a bit with two input Boolean variables *U* and *V* produces a result of 1 or 0. The logic circuitry of parallel OR on one-bit is shown in Figure 3. The corresponding truth table of the one-bit OR is shown in Table 3.

Table 3: The truth table of the one-bit OR

Input		Output
U_k	V_k	$OR_k = U_k \vee V_k$
0	0	0
0	1	1
1	0	1
1	1	1

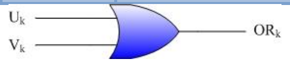


Figure 3: Logic circuitry of parallel OR on one bit

```
**ParallelOneBitOR**($T_0$, $U_k$, $V_k$, $OR_k$)
$T_1^{U=1} = +(T_0, U_k^1)$ and $T_1^{U=0} = -(T_0, U_k^1)$.
$T_2^{U=1,V=1} = +(T_1^{U=1}, V_k^1)$ and $T_2^{U=1,V=0} = -(T_1^{U=1}, V_k^1)$
$T_2^{U=0,V=1} = +(T_1^{U=0}, V_k^1)$ and $T_2^{U=0,V=0} = -(T_1^{U=0}, V_k^1)$
If (Detect($T_2^{U=1,V=1}$) = = "yes") then
    Append-head($T_2^{U=1,V=1}$, $OR_k^1$)  EndIf
If (Detect($T_2^{U=1,V=0}$) = = "yes") then
    Append-head($T_2^{U=1,V=0}$, $OR_k^1$)  EndIf
If (Detect($T_2^{U=0,V=1}$) = = "yes") then
     Append-head($T_2^{U=0,V=1}$, $OR_k^1$)  EndIf
If (Detect($T_2^{U=0,V=0}$) = = "yes") then
     Append-head($T_2^{U=0,V=0}$, $OR_k^0$)  EndIf
$T_0 = \cup(T_2^{U=1,V=1}, T_2^{U=1,V=0}, T_2^{U=0,V=1}, T_2^{U=0,V=0})$
EndAlgorithm
```

Figure 4: Parallel OR operation on one bit algorithm

5.3 XOR Operation on Bioinformatics Computing

The Exclusive-OR (XOR) operation of a bit with two input Boolean variables *U* and *V* generates an output of 1 or 0. The logic circuitry of parallel XOR on one-bit is shown in Figure 5. The corresponding truth table of the one-bit XOR is shown in Table 4:

Table 4: The truth table of the one-bit XOR

Input		Output
U_k	V_k	$XOR_k = U_k \oplus V_k$
0	0	0
0	1	1
1	0	1
1	1	0

Figure 5: Logic circuitry of Parallel XOR on one bit

```
**ParallelOneBitXOR**($T_0$, $U_k$, $V_k$, $XOR_k$)
$T_1^{U=1} = +(T_0, U_k^1)$ and $T_1^{U=0} = -(T_0, U_k^1)$.
$T_2^{U=1,V=1} = +(T_1^{U=1}, V_k^1)$ and $T_2^{U=1,V=0} = -(T_1^{U=1}, V_k^1)$
$T_2^{U=0,V=1} = +(T_1^{U=0}, V_k^1)$ and $T_2^{U=0,V=0} = -(T_1^{U=0}, V_k^1)$
If (Detect($T_2^{U=1,V=1}$) = = "yes") then
    Append-head($T_2^{U=1,V=1}$, $XOR_k^0$)  EndIf
If (Detect($T_2^{U=1,V=0}$) = = "yes") then
```

SEEK DIGITAL LIBRARY

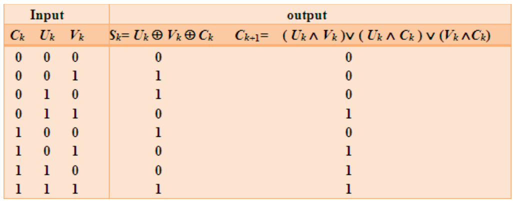
```
    Append-head($T_2^{U=1,V=0}$, $XOR_k^1$)  EndIf
If (Detect($T_2^{U=0,V=1}$) = = “yes”) then
    Append-head($T_2^{U=0,V=1}$, $XOR_k^1$)  EndIf
If (Detect($T_2^{U=0,V=0}$) = = “yes”) then
    Append-head($T_2^{U=0,V=0}$, $XOR_k^0$)  EndIf
$T_0 = \cup(T_2^{U=1,V=1}, T_2^{U=1,V=0}, T_2^{U=0,V=1}, T_2^{U=0,V=0})$.
EndAlgorithm
```

Figure 6: Parallel XOR operation on one bit algorithm

5.4 Bio-Arithmetic Parallel Adder on One Bit

A one-bit adder has three inputs and two outputs. The logic circuitry of parallel adder on one-bit is shown in Figure 7, and the truth table of the one-bit adder is shown in Table 5.

Table 5: The truth table of the one-bit adder



Input			output	
C_k	U_k	V_k	$S_k = U_k \oplus V_k \oplus C_k$	$C_{k+1} = (U_k \wedge V_k) \vee (U_k \wedge C_k) \vee (V_k \wedge C_k)$
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

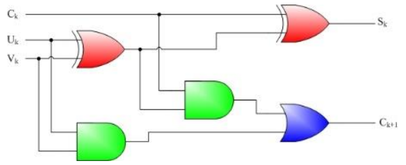


Figure7: Logic circuitry of parallel adder on one bit

Based upon the logic circuitry in Figure 7, we can derive the bio-algorithm of parallel adder on one-bit in Figure 8.

```
ParallelOneBitAdder($T_0$, $U_k$ , $V_k$, $C_k$)
ParallelOneBitXOR($T_0$, $U_k$,$V_k$, $XOR_k$)
ParallelOneBitXOR($T_0$, $XOR_k$, $C_k$ , $S_k$)
ParallelOneBitAND($T_0$, $U_k$ ,$V_k$ , $AND_k^1$)
ParallelOneBitAND($T_0$, $C_k$ ,$V_k$ , $AND_k^2$)
ParallelOneBitAND($T_0$, $U_k$ , $C_k$ , $AND_k^3$)
ParallelOneBitOR($T_0$, $AND_k^1$, $AND_k^2$, $OR_k^1$)
ParallelOneBitOR($T_0$, $OR_k^1$, $AND_k^3$, $OR_k^2$)
$T_1 = +(T_0, OR_k^{2^1})$ and $T_2 = -(T_0, OR_k^{2^0})$
If(Detect($T_1$)==”yes”) then
        Append-head ($T_1$,$C_{k+1}^1$)    EndIf
If(Detect($T_2$)==”yes”) then
        Append-head ($T_2$,$C_{k+1}^0$)  EndIf
$T_0 = \cup(T_1, T_2)$
EndAlgorithm
```

Figure 8: Parallel adder on one-bit algorithm

5.5 Bio-Arithmetic Parallel Adder on n Bits

In this section, we use the bio-arithmetic adder on one-bit to construct the Parallel Adder in Figure 9.

```
ParallelAdder($T_0$, U, V, n)
Append($T_0$, $C_1^0$)
For k=1 to n
    ParallelOneBitAdder($T_0$, $U_k$ , $V_k$, $C_k$ )
EndFor
EndAlgorithm
```

Figure 9: Parallel adder algorithm

5.6 Bio-Arithmetic Parallel Comparator on One Bit

The following algorithm is applied to compare the stickers from tubes T_a and T_b. Tube $T_0^=$ is the first parameter containing equal comparisons and to pass these equal comparisons to algorithm Parallel Comparator ($T_0^{EDGE\_temp}$, $T_0^{overlay}$, T_a, T_b, m, n, g, b) in Figure 11. Algorithm for parallel execution on a one bit comparison is shown in Figure 10.

```
OneBitComparator($T_0^=$, $T_a$, $T_b$, $p$ ,$d$)
$T_1^{1st\_on} = +(T_a, s_{p,1}^1)$and $T_1^{1st\_off} = -(T_a, s_{p,1}^1)$
$T_2^{2nd\_on} = +(T_a, s_{p,2}^1)$ and $T_2^{2nd\_off} = -(T_a, s_{p,2}^1)$
$T_3^{1st\_on} = +(T_b, s_{d,1}^1)$ and $T_3^{1st\_off} = -(T_b, s_{d,1}^1)$
$T_4^{2nd\_on} = +(T_b, s_{d,2}^1)$ and $T_4^{2nd\_off} = -(T_b, s_{d,2}^1)$
If (Detect($T_1^{1st\_on}$)=‘yes’ and Detect($T_3^{1st\_on}$)=‘yes’) then
    If(Detect($T_2^{2nd\_on}$)=‘yes’ and Detect($T_4^{2nd\_on}$)=‘yes’) then
        $T_0^= = \cup(T_0^=, T_1^{1st\_on}, T_3^{1st\_on}, T_2^{2nd\_on}, T_4^{2nd\_on})$  EndIf   EndIf
If (Detect($T_1^{1st\_on}$) = ‘yes’ and Detect($T_3^{1st\_on}$) = ‘yes’) then
    If(Detect($T_2^{2nd\_off}$)=‘yes’ and Detect($T_4^{2nd\_off}$)=‘yes’) then
        $T_0^= = \cup(T_0^=, T_1^{1st\_on}, T_3^{1st\_on}, T_2^{2nd\_off}, T_4^{2nd\_off})$  EndIf   EndIf
If(Detect($T_1^{1st\_off}$) = ‘yes’ and Detect($T_3^{1st\_off}$) = ‘yes’) then
    If(Detect($T_2^{2nd\_on}$)=‘yes’ and Detect($T_4^{2nd\_on}$)=‘yes’) then
        $T_0^= = \cup(T_0^=, T_1^{1st\_off}, T_3^{1st\_off}, T_2^{2nd\_on}, T_4^{2nd\_on})$  EndIf   EndIf
If(Detect($T_1^{1st\_off}$) = ‘yes’ and Detect($T_3^{1st\_off}$) = ‘yes’) then
    If(Detect($T_2^{2nd\_off}$)=‘yes’ and Detect($T_4^{2nd\_off}$)=‘yes’) then
        $T_0^= = \cup(T_0^=, T_1^{1st\_off}, T_3^{1st\_off}, T_2^{2nd\_off}, T_4^{2nd\_off})$  EndIf  EndIf
EndAlgorithm
```

Figure 10: Parallel comparator on one bit

5.7 Bio-arithmetic Parallel Comparator on n Bits

The following algorithm, **ParallelComparator** (T_0, T_0 *overlay*, T_a, T_b, m, n, g, b), is an n-bit comparator. Parallel execution on n bit comparisons is shown in Figure 11.

```
ParallelComparator($T_0$, $T_0^{overlay}$ , $T_a$ , $T_b$ , $m$, $n$, $g$, $b$)
For $d$ =0 to Min($n$-$m$,$b$-$g$)
    For $p$=$n$ downto $m$
        OneBitComparator($T_0^=$, $T_a$, $T_b$, $p$, $g+d$)
        If (Detect($T_0^=$)=“yes”) then
            Append($T_0^{overlay}$,$O_{p,g+d}^1$)
            Discard($T_0^=$)  EndIf
    EndFor
EndFor
If (Detect($T_0^{overlay}$)=“yes”) then
    $T_0 = \cup(T_0, T_0^{overlay})$  EndIf
Discard($T_0^{overlay}$)
EndAlgorithm
```

Figure 11: parallel comparator om n bits

6 Proposed A Fast Bioinformatics Approach for Solving Backtracking of DNA Sequence Evolution in One Dimensional Cellular Automata

In this research, the entire bioinformatics approach for solving backtracking of DNA sequence evolution in one dimensional cellular automata is accomplished by algorithms I and II. They are DNA sequemce evolution in one dimensional automata and backtracking of DNA sequence evolution repectively.

```
Algorithm :SolvingCAModelforDNAEvolutionBacktracking
  (a)Algorithm I : DNASequemceEvolutionInOneDimentionalCA
  (b)Algorithm II: BacktrackingofDNASequenceEvolution
ENDAlgorithm
```

Figure 12: Proposed algorithms to Construct and backtracking of DNA Sequence Evolution in one dimensional CA.

6.1 Proposed Bioinformatics Algorithms to Construct DNA Sequence Evolution

Based on each evolved procedure, the inputs are the rule matrix and the CA status in the previous evolution step. Once we get the newest status for the current step, record it and proceed to the next step until the last step *f* is completed.

In algorithm I, there are several procedures proposed to solve the construction of the DNA sequence evolution model in one dimensional cellular automata.

```
(a)Algorithm I: DNASequemceEvolutionInOneDimensionalCA
    For cellular automaton step=0 to $f$
      (a1) ConstructRuleMatrix ($T_0^{Rule}, T_1^{Rule}, \ldots, T_{n-1}^{Rule}, Rule$)
      (a2) InputCAStatus($T_{temp}^{sequence}, T_{latest}^{sequence}, T_{initial}^{sequence}$)
      (a3) ExecuteCAStatus($T_{latest}^{sequence}, T_{temp}^{sequence}, T_0^{Rule}, T_1^{Rule}, \ldots, T_{n-1}^{Rule}$)
```

SEEK DIGITAL LIBRARY

Proc. of the Intl. Conf. on Advances In Bio-Informatics And Environmental Engineering - ICABEE 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-019-4 doi: 10.15224/ 978-1-63248-019-4-03

```
            (a4) SaveCAStatus($T^{sequence}_{record}$ ,$T^{sequence}_{latest}$)
    EndFor
EndAlgorithm
```

Figure 15: Algorithms to backtracking of DNA sequence evolution in one dimensional cellular automata.

6.1.1 One Dimensional Cellular Automata

Cellular automata were proposed by von Neumann and Ulam. Any system with many identical discrete elements undergoing deterministic local interactions may be modeled as cellular automata.

Cellular automaton usually has several grids of elements which each has the finite number of states, such as “on” or “off”, “has something” or nothing. The number of grids can be any finite number or dimensions for the need of any researchers. Therefore, the presentation example of the one dimensional cellular automaton can be shown in Figure 12:

Figure 12: Example of one dimensional cellular automaton

Suppose that the initial configuration of the one dimensional cellular automaton is shown in Figure 12. Using rule 90 [2] on the initial configuration, the five following continous steps in Figures (a) to (e) for the one dimensional cellular automaton evolution are shown in Figure 13.

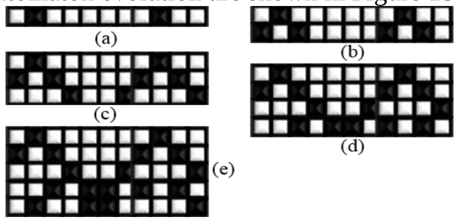


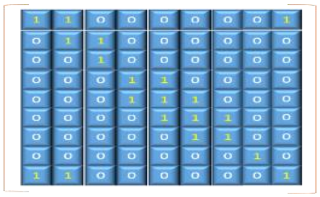
Figure 13: The five results of the one dimensional cellular automaton evolved from Figure 12.

6.1.2 Construction of DNA Sequence Evolution

Figure 14 shows the construction of rule matrix. Suppose that DNA sequence is n bit length, and the width of each one dimensional cellular automaton is n. The following algorithm constructs the rule matrix, which the matrix is $n*n$.

```
Procedure ConstructRuleMatrix($T^{Rule}_0, T^{Rule}_1, \ldots, T^{Rule}_{n-1}, Rule$)
      (1)For $i$ =0 to $n$-1
         (1a) For $j$ =0 to $n$-1
               (1aa) Append ($T^{Rule}_i$,$ith\ row\ of\ rule\ matrix$).
         EndFor
      EndFor
EndProcedure
```

Figure 14: Construction of the rule matrix.



1	1	0	0	0	0	0	0	1
0	1	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0
0	0	0	0	1	1	1	0	0
0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	1	0
1	1	0	0	0	0	0	0	1

Figure 15: Example of rule matrix constructed by Figure 13

Figure 16 shows the CA status used for the next configuration. Suppose that DNA sequence is n bit length, and the width of next one dimensional cellular automaton is n.

```
Procedure InputCAStatus ($T^{sequence}_{temp}$ ,$T^{sequence}_{latest}$ ,$T^{sequence}_{initial}$)
         (1)Discard($T^{sequence}_{temp}$)
         (2)If cellular automaton step=0
      (2a)Append($T^{sequence}_{temp}$ , $T^{sequence}_{latest}$)
    Else
      (2b)Append($T^{sequence}_{temp}$ , $T^{sequence}_{initial}$)
EndProcedure
```

Figure 16: CA status for the next configuration

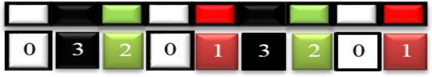
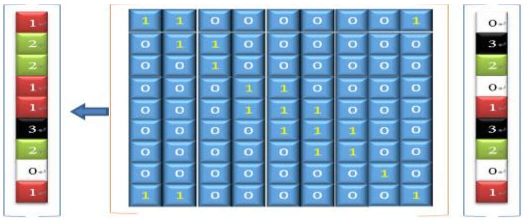


Figure 17: CA status outcomes generated from Figure 16.

The next status of one-dimensional cellular automaton for the newest configuration can be used in Figure 18.

```
Procedure ExecuteCAStatus ($T^{sequence}_{latest}$ , $T^{sequence}_{temp}$ , $T^{Rule}_0$ , $T^{Rule}_1$ , … , $T^{Rule}_{n-1}$)
   (1)Discard($T^{sequence}_{latest}$)
   (2) For i = 0 to $n$-1
     (2a)For j= 0 to $n$-1
        (2aa)Discard($T^{Execution}_{temp}$)
        (2ab)Discard($T^{Execution}_{temp2}$)
        (2ac)ParallelMultiplier($T^{Execution}_{temp}$, U,V, , $T^{Rule}_i$, $T^{sequence}_{temp}$)
        (2ad)ParallelAdder($T^{Execution}_{temp2}$, $T^{Execution}_{temp2}$ , $T^{Execution}_{temp}$)
     EndFor
     (2b)Append($T^{sequence}_{latest}$ , $T^{Execution}_{temp2}$)
   EndFor
EndProcedure
```

Figure 18: Execution of one dimensional cellular automaton.



1	1	0	0	0	0	0	0	1
0	1	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	1	1	0	0	0	0
0	0	0	1	1	1	0	0	0
0	0	0	0	1	1	1	0	0
0	0	0	0	0	1	1	0	0
0	0	0	0	0	0	0	1	0
1	1	0	0	0	0	0	0	1

Figure 19: Execution example of Figure 18.

The result of one dimensional cellular automaton new status for a new configuration is stored in Figure 20.

```
Procedure SaveCAStatus($T^{sequence}_{record}$ ,$T^{sequence}_{latest}$)
      (1)Append($T^{sequence}_{record}$ ,$T^{sequence}_{latest}$)
EndProcedure
```

Figure 20: Storage to record one dimensional CA new status.

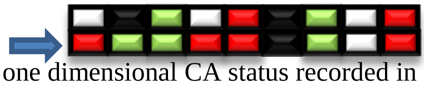


Figure 21: one dimensional CA status recorded in Figure 20.

After storing our newest status of CA, then proceed to next CA step and back to **procedure (a1)** in Algorithm I for the next step execution until all step executions are completed. The Figure22 shows the CA status after five step executions are completed.

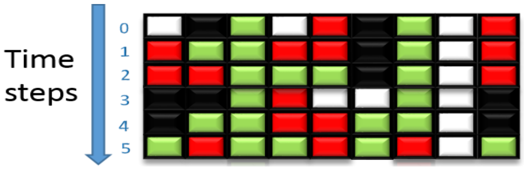


Figure 22: Example of one dimensional cellular automaton status after five step executions are completed.

6.2 Proposed Bioinformatics Algorithms for Backtracking of DNA Sequence Evolution

Based on the evolved rule, each configuration of width n in one dimensional cellular automaton evolves to its next CA status. As the section 3 mentioned, the order finding problem can be identified as finding the order of one of the prime factor for an integer.

In algorithm II, we proposed several procedures shown in Figure 23 to solve the backtracking of the DNA sequence evolution in one dimensional CA.

```
(b)Algorithm II: BacktrackingofDNASequenceEvolution
```

SEEK DIGITAL LIBRARY

```
(b1) CreateSolutionspace($T_0, T_1, \ldots, T_{N-1}, Group\ N$)
(b2) CalculategcdforSolutionSpace($T_{com}, T_{goal}, T_{temp}, T_0, T_1, \ldots, T_{N-1}$)
(b3) CalculateModforSolutionSpace
( $T_{com}, T_{temp}, T_0, T_1, \ldots, T_{N-1}, T_{\mathbf{remainder}}, T_r$)
(b4)JudgeorderforSolutionSpace
($T_0, T_1, \ldots, T_{N-1}, T_{goal}, T_{temp\_r}, T_{temp\_a}, T_{temp\_log}, T_{temp\_result}, T_{answer}$ )
  EndAlgorithm
```

Figure 23: Algorithms for solving backtracking of the DNA sequence evolution in one dimensional CA.

6.2.1 Order-Finding

The backtracking of the tumor growth in *reversible* one dimensional cellular automaton is used to find out its final configuration that can be evolved to a specific or initial configuration. Suppose a function, $\theta: \{x|0 \leq x \leq 4^n - 1\} \rightarrow \{y|0 \leq y \leq 4^m - 1\}$, is called a one way function. Lemma 1 shows that a *reversible* one dimensional cellular automaton is a one way function so that one dimensional cellular automaton is an one to one relationship. Hence, order finding can be used to solve backtrcking of one dimensional cellular automaton.

Lemma 1: A *reversible* one dimensional cellular automaton is a one way function.

Suppose that if a, b, and N are integers with N ≥ 1 and a ≡ b(mod N). Then we let Z_N denote the set Z_N = {0,…,N-1}.If in addition N is prime, then Z_N forms a field. We write Z_N^* to denote the following set :

$$Z_N^* = \{a \in Z_N : \gcd(a, N) = 1\}$$

For any element a ∈ Z_N^* there exists a unique element b ∈ Z_N^* that satisfies

$$ab \equiv 1(\text{mod N}) \text{ and } b = a^{-1}(\text{mod N})$$

Now, for a given element a ∈ Z_N^*, the *order of* a *in* Z_N^*(or the *order of* a *modulo* N) is the smallest positive integer r of *n* =$\lceil \log_2(N * N) \rceil$ bits such that

$$a^r \equiv 1(mod\ N)$$

and $0 \leq r \leq 4^n - 1$. Assume that a system has 4^n possible configurations in which it includes the first function *F*: $\{k|0 \leq k \leq 4^n - 1\} \rightarrow \{0, 1\}$, and the second function

$$G: \{k|0 \leq k \leq 4^n - 1\} \rightarrow \{a^k \equiv 1(mod\ N)\}$$

The relationships between functions F and G are shown in Table 5. In a one dimensional cellular automaton with *n* cells, its evolved function is $\theta: \{k|0 \leq k \leq 4^n - 1\} \rightarrow \{v|0 \leq v \leq 4^m - 1\}$, where $\{k|0 \leq k \leq 4^n - 1\}$ is a set of all of the initial configurations and $\{v|0 \leq v \leq 4^m - 1\}$ is a set of all of the evolved configurations. If function *F* finds the corresponding initial configuration *k* for *v* = θ (*k*), then *F*(*k*) ∊ {1}. Otherwise, *F*(*k*) ∊ {0}.

Table 5: A relation of degree 2, *R*.

F(k)	$G(\mathbf{k}) = \boldsymbol{a^k} \equiv \mathbf{1}(\boldsymbol{mod\ N})$
F(0)	$G(0) = a^0 \equiv 1(mod\ N)$
⋮	⋮
F(512)	$G(512) = a^{512} \equiv 1(mod\ N)$
⋮	⋮
F(4^n)	$G(4^n) = a^{4^n} \equiv 1(mod\ N)$

6.2.2 Backtracking of DNA Sequence Evolution

Suppose that DNA sequence is n bit length, then there are 4^n possible configurations. Procedure **CreateSolutionspace** constructs the solution space of the group of integer N, which we named as ℤN and we denote the set ℤN = {0,…,N-1} is corresponding to n evolution steps of DNA sequence in one dimensional CAthat is used for order finding.

```
Procedure CreateSolutionspace ($T_0, T_1, \ldots, T_{N-1}, Group\ N$)
   (1)For f =0 to N-1
      (1a) Append ($T_f$, $fth$ Group element).
   EndFor
EndProcedure
```

Figure 24: Create the solution space for order finding

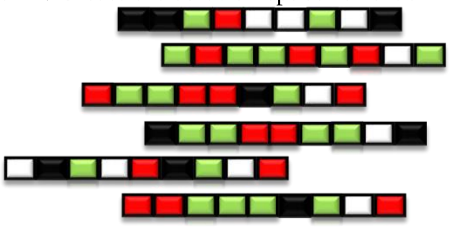


Figure 25:Example of creating the solution space

Based upon the one way function, suppose that*"A"* is an integer and co-prime with integer "N", and we let ℤN denote the set ℤN = {0,…,N-1}, then any possible candidate "*A*" can be defined as: $\{A \in Z_N : \gcd(A, N) = 1\}$ After procedure **CalculategcdforSolutionSpace** filterates all possible configurations, one possible solution can be found if $\gcd(A, N) = 1$.

In order to compare the values of two tubes in order finding, Function **ParallelComparator** is modified in the following:

```
ParallelComparator($T_R^=$, $T_R^<$, $T_R^>$,$T_A$ ,$T_B$, n)
   (1) For k = n downto 1
      (2) ParallelOneBitComparator ($T_C^=$, $T_C^<$, $T_C^>$ , $T_A$ ,$T_B$)
         (2a) If (Detect($T_C^=$) = = "yes") then
               $T_R^=$ = ∪( $T_R^=$, $T_C^=$).
         (2b) Else If (Detect($T_C^<$) = = "yes") then
               $T_R^<$ = ∪( $T_R^<$, $T_C^<$).
         (2c) Else If (Detect($T_C^>$) = = "yes") then
               $T_R^>$ = ∪( $T_R^>$, $T_C^>$).
         EndIf
   EndFor
EndAlgorithm
```

Figure 26: Modified parallel comparator

```
Procedure CalculategcdforSolutionSpace
($T_{com}, T_{goal}, T_{temp}, T_0, T_1, \ldots, T_{N-1}$)
   (1)Append($T_{com}$, 1)
   (2)Append($T_{goal}$, 0)
   (3)For f=1 to N-1
      (3a)Append ($T_{temp}$,N)
      (3b)Repeat
         (3ba)ParallelComparator($T^=_{compare}$, $T^<_{compare}$, $T^>_{compare}$, $T_{temp}$, $T_f$ , n)
         (3bb)If (Detect($T^>_{compare}$)="Yes")
               (3bba) ParallelSubtractor($T_{temp}$, $T_{temp}$, $T_f$, n)
            Else
               (3bbb) ParallelSubtractor($T_f$, $T_f$, $T_{temp}$, n)
         (3bc)ParallelComparator( $T^=_{compare2}$ , $T^<_{compare2}$ , $T^>_{compare2}$ , $T_{temp}$, $T_{com}$ , n)
         (3bd)ParallelComparator( $T^=_{compare3}$ , $T^<_{compare3}$ , $T^>_{compare3}$ , $T_{temp}$, $T_{goal}$ , n)
         (3be)ParallelComparator($T^=_{compare4}$, $T^<_{compare4}$, $T^>_{compare4}$, $T_f$, $T_{com}$ , n)
         (3bf)ParallelComparator($T^=_{compare5}$, $T^<_{compare5}$, $T^>_{compare5}$, $T_f$, $T_{goal}$ , n)
      Until(Detect($T^=_{compare2}$)="Yes" || Detect($T^=_{compare3}$)="Yes" ||
       Detect($T^=_{compare4}$)="Yes"|| Detect($T^=_{compare5}$)="Yes")
      (3c)If(Detect($T^=_{compare3}$)="Yes"||Detect($T^=_{compare5}$)="Yes")
         (3ca)Discard($T_f$)
      Else
         (3cb)Discard($T_f$)
         (3cc)Append($T_f$, $fth$ Group element )
      (3d) Discard($T_{temp}$)
   EndFor
   EndProcedure
```

Figure 27: Calculation for the greatest common divisor

SEEK DIGITAL LIBRARY

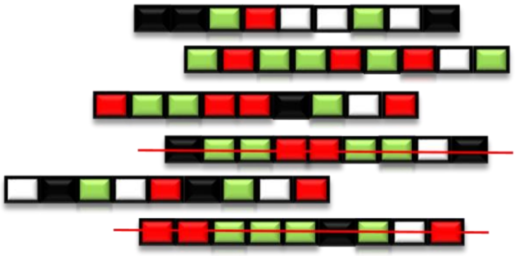


Figure 28: Example of $\gcd(a, N) = 1$ for filtration

Figure 28 shows using $\gcd(a, N) = 1$tofiltrate all possible non-prime DNA sequences. In Figure 29, we can find the exponent "r" in the formula which satisfies $a^r \equiv 1(mod\ N)$. If the value of "a" is not prime, then that value is are filtrated. We expect that the value of $1\ mod\ N$ is 1. In order to find the value of "r", the result of $a^r mod\ N$ must be equal to 1.

```
Procedure CalculateModforSolutionSpace
( $T_{com}$, $T_{temp}$, $T_0$, $T_1$,…, $T_{N-1}$, $T_{\mathbf{remainder}}$, $T_r$)
        (1)For f = 0 to N-1
          (1a)If (Detect($T_f$)=”No”)
               Terminate and go to the next loop
          (1b)Append ($T_{temp}$,N)
          (1c)For k = 0 to 4^n
               (1ca) ParallelModular($T_{\mathbf{remainder}}$, $T_f$, $T_{temp}$,n)
               (1cb)ParallelComparator($T^{=}_{compare6}$, $T^{<}_{compare6}$, $T^{>}_{compare6}$, $T_{\mathbf{remainder}}$, $T_{com}$ , n)
              (1cc)If(Detect($T^{>}_{compare6}$) = "Yes")
                   (1cca) ParallelMultiplier($T_f$, $T_f$, $T_f$, a, b )
                   (1ccb) ParallelAdder($T_r$, $T_r$, 1, n)
                   (1ccc)Discard($T_{\mathbf{remainder}}$)
               Else if(Detect($T^{=}_{compare6}$) = "Yes")
                    Terminate this loop
          EndFor
        (1d) Append-Head ($T_f$,$T_r$)
        (1e)Discard($T_{temp}$)
     EndFor
EndProcedure
```

Figure 29: Calculation for the Modulo Function

Once we find all of values of "r" for the corresponding group elements, the optimal solution "r" is found, which satisfies function: $(\mathrm{r} \log a) mod\ N = \log 1$, where $\log 1 = 0$.

```
Procedure JudgeorderforSolutionSpace
($T_0$, $T_1$,…, $T_{N-1}$, $T_{goal}$, $T_{temp\_r}$, $T_{temp\_a}$, $T_{temp\_log}$, $T_{temp\_result}$, $T_{answer}$ )
        (1)For f=0 to N-1
        (1a)If (Detect($T_f$)=”No”)
             Terminate and go to the next loop
         (1b)Append ($T_{temp}$,N)
         (1c) $T_{temp\_r}$ =+($T_f$, $r$)
          (1d) $T_{temp\_a}$ =-($T_f$, $r$)
         (1e)log($T_{temp\_a}$,$T_{temp\_a}$)
          (1f) ParallelMultiplier($T_{temp\_log}$, $T_{temp\_a}$, $T_{temp\_r}$, a, b )
          (1g) ParallelModular($T_{temp\_result}$, $T_{temp\_log}$,$T_{temp}$, n)
          (1h)ParallelComparator( $T^{=}_{compare7}$ , $T^{<}_{compare7}$ , $T^{>}_{compare7}$ , $T_{temp\_result}$, $T_{goal}$ , n)
          (1i)If (Detect($T^{=}_{compare7}$)=”Yes”)
             (1ia)Append($T_{answer}$, $T_{temp\_r}$)
             Else
              (1ib) Discard($T_{log}$)
              (1ic) Discard($T_{temp\_r}$)
              (1id) Discard($T_{temp\_a}$)
              (1ie) Discard($T_{temp\_result}$)
              (1if) Discard($T_{temp}$)
      EndFor
EndProcedure
```

Figure 30 : Backtracking algorithm after judgments for all possible selections

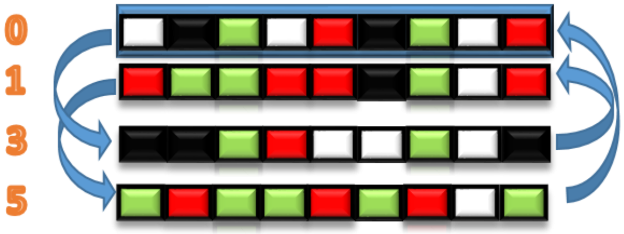


Figure 31: Example of backtracking to the original intial configuration after judgments for all possible selections.

Figure 31 shows number 0 to be the original intial configuration after judgments for all possible selections by using function: $(\mathrm{r} \log a) mod\ N = \log 1$.

7 Complexity for Solving Backtracking of DNA Sequence Evolution

(1) The time complexity of proposed optimal bioinformatics algorithm (**Algorithm I**) to construct DNA sequence evolution in one dimensional cellular automata is found in $O(n^2)$ polynomial bound.

(2) The time complexity of proposed optimal bioinformatics algorithm (**Algorithm II**) for the backtracking of the DNA sequence evolution is in $\mathrm{O}(\mathrm{n}^2)$ polynomial bound.

8 Conclusion

It is a well-known fact that the DNA mutation plays a very important role in DNA sequence evolution. The Backtracking problem of DNA sequence evolution in one dimensional cellular automaton has ben recognized as a NP problem. In order to solve backtracking problem, a newly developed optimal bioinformatics algorithm for solving a backtracking of the DNA sequence evolution is proposed. First, one dimensional cellular automaton is used to construct a bioinformatics graphical DNA sequence evolution. Second, a bioinformatic order-finding algortihm solves backtracking of the DNA sequence evolution. With bioinformatics computing which fully utilizing massive storage and parallel computations, the construction of DNA sequence evolution and its backtracking have become more efficient and more faster.

9 References

[1] Ch. Mizas , G.Ch. Sirakoulis , V. Mardiris , I. Karafyllidis , N. Glykos , R. Sandaltzopoulos(2008), "*Reconstruction of DNA sequences using genetic algorithms and cellular automata: Towards mutation prediction?*", BioSystems Vol.92, pp.61 – 68.

[2]Wolfram, Stephen (1983). "*Statistical Mechanics of Cellular Automata*".Reviews of Modern Physics Michael (Shan-Hui) Ho Vol.55/3, pp. 601–644.

[3] Shor P. W. (1994), "*Algorithm for quantum computation: discrete logarithm and factoring algorithm.*" Proceedings of the 35th Annual IEEE Symposium on Foundation of Computer Science :pp.124-134.

[4] Michael (Shan-Hui) Ho, W.L Chang, Minyi Guo (2007), "*Application of Bio-molecular Computing to Breakthrough in Cryptography.*" Systems Bioinformatics : An Engineering Case-Based Approach : pp.319-337.

SEEK
DIGITAL LIBRARY