Pseudorandom Pattern Generation by a 4-Neighborhood Cellular Automata Based on a Probabilistic Analysis

Abhishek Seth, S. Bandyopadhyay, U. Maulik. [*]

Abstract—**In this article, we construct a random number generator using a one dimensional, non-uniform 4-Neighborhood Cellular Automata (4NCA). A probabilistic analysis of CA rules has been done to select the appropriate rules for 4NCA random number generator (RNG). A comparison is made between the pseudo random patterns generated by using the proposed CA RNGs with those obtained using a recent Cellular Programming (CP) evolved CA RNGs. The results show that our approach outperforms CP both in terms of average time taken to evolve CA rules and in terms of quality of pseudo random patterns generated. The proposed approach is also shown to be better than the common generators such as Shift Register, Congruential Generator and Lagged Fibonacci Generator.**

Keywords: 4NCA, Entropy, Cellular Programming, Probabilistic Analysis.

1 Introduction

Cellular Automata (CA) has been used for pseudorandom number generation in the past [4]. They are also used to implement random number generators (RNGs) in cryptographic devices [6] and in Built-In-Self-Test (BIST) circuits. With the increase in the computational capabilities of computers, the demand of RNGs have likewise increased [7] to carry out more sophisticated simulations. Wolfram [14], in 1986, suggested that CA could be used for efficient hardware implementation of random number generation due to their simplicity and regularity of design.

One dimensional CA based RNGs have been extensively studied in past [4] and their superiority over other widely used methods such as linear feedback shift registers (LFSRs) has been convincingly established, especially in the case of delay type faults which require patterns in specific order [3]. CA based RNGs can also be evolved automatically by genetic algorithms. Sipper and Tomassine [10] described a process of evolving the CA truth table using a co-evolution process termed Cellular Programming.

To provide a standardized means of comparing random number quality among CA based RNGs, Tomassine et. al. introduced the use of Marsaglia's highly regarded Diehard random number test suite. We also use this acknowledged test suite for evaluating the performance of our random number generation algorithm.

In this paper we propose a new random number generation method for 4 neighborhood CA based on a probabilistic analysis of the CA rules. A comparison is made between patterns generated by some standard RNGs, CP evolved RNGs and our proposed RNG in terms of Marsaglia's Diehard random number test suite introduced by Tomassine et al. as a means of comparing random number quality [12]. It is shown that the proposed method produce patterns that are better in terms of their randomness as compared to the other techniques.

2 CA Preliminaries

Cellular automata can be thought of as dynamical systems, discrete in both time and space [13]. It consists of an infinite, regular grid of cells, each in one of a finite number of states. The grid can be in any finite number of dimensions. Time is also discrete and the state of a cell at time t is a function of the states of a finite number of cells (called its neighborhood) at time $t-1$. These neighbors are a selection of cells relative to the specified cell, and do not change. Here, we will only consider Boolean automata in which the cellular state, $s \in \{0,1\}$.

All cells update its value synchronously in discrete time steps accordingly to some rule R. Such rule is based on the state of the cell itself and the state of r of its neighbors:

$$s^{j+1}{}_i = R(s^j{}_{i-r}, ..., s^j{}_{i-1}, s^j{}_i, s^j{}_{i+1}, ..., s^j{}_{i+r})$$

where $s^{j+1}{}_i$ is a value of i^{th} cell (the state of a cell) in step j and r is a radius of the neighborhood. Neighborhood is composed of $m = 2*r+1$ cells. This makes $n = 2^m$

[*]Abhishek Seth is student at Department of Computer Science and Engineering, Indian School of Mines, Dhanbad. Email: abhishekseth8887@gmail.com. S. Bandyopadhyay is Senior Member, IEEE and faculty at Machine Intelligence Unit Indian Statistical Institute Kolkata. Email: sanghami@isical.ac.in. U. Maulik is Senior Member, IEEE and faculty at Department of Computer Science and Engineering, Jadavpur University, Kolkata. Email: drumaulik@cse.jdvu.ac.in

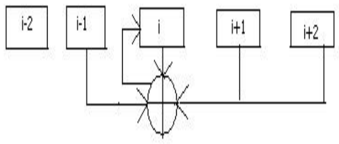

Figure 1: 4NCA with two right neighbors

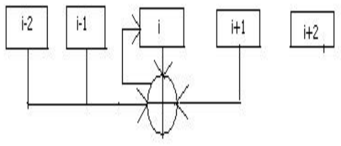

Figure 2: 4NCA with two left neighbors

possible configurations of that neighborhood [8]. Rules are usually named using standard convention.

A CA characterized by EXOR and/or EXNOR dependence is called an additive CA [4]. If in a CA the neighborhood dependence is EXOR, then it is called a non complemented rule. For neighborhood dependence of EXNOR (where there is an inversion of the modulo-2 logic), the CA is called a complemented CA. The corresponding rule involving the EXNOR function is called a complemented rule. If in a CA same rule applies to all cells, then the CA is called a uniform CA; otherwise the CA is called a hybrid CA. There can be various boundary conditions; namely, null (where extreme cells are connected to logic '0'), periodic (extreme cells are adjacent) etc [4]. Nonuniform, or inhomogeneous, cellular automata function in the same way as uniform ones, the only difference being in the cellular rules that need not be identical for all cells. Nonuniform CAs share the basic "attractive" properties of uniform ones: simplicity, parallelism and locality [9].

2.1 4-Neighborhood Cellular Automata

4NCA is special type of CA in which the state of cell i at time $t+1$ depends on states of cells $i-2$, $i-1$, i and $i+1$ or cells $i-1$, i, $i+1$ and $i+2$ at time t [1].

$$\begin{aligned} q[i](t+1) = \; & f(q[i-2](t), q[i-1](t), q[i](t), \\ & q[i+1](t), q[i+2](t)) \end{aligned}$$

where f defines the CA rules.

The 4NCA consist of an array of identical memory cells arranged in one dimensional fashion. In 4NCA a cell depends on 4 of it's neighbors. This is done by adding one more neighbor either from left or from right but not both. Each cell may have either of the following neighborhood dependencies:
(1) Two left neighbors, self and one right neighbor dependency.

$$q[i](t+1) = f(q[i-2](t), q[i-1](t), q[i](t), q[i+1](t)).$$

(2) Two right neighbors, self and one left neighbor dependency

$$q[i](t+1) = f(q[i-1](t), q[i](t), q[i+1](t), q[i+2](t)).$$

The proposed model of cellular automata may use both the above dependencies interchangeably but the left of the immediate left and right of the immediate right cells of cell i must not be connected to cell i at the same time. As a consequence though it is not configured in this mode, it can exploit 5 neighborhood dependency.

In this paper we use CA having additive rule. An additive CA rule can be defined by following equation:

$$\begin{aligned} q[i](t+1) = \; & M \oplus P \bullet q[i-2](t) \oplus Q \bullet q[i-1](t) \\ & \oplus R \bullet q[i](t) \oplus S \bullet q[i+1](t) \\ & \oplus T \bullet q[i+2](t) \end{aligned}$$

$q[i](t)$ means the state of cell i at time t.
$M = 0$; indicates linear additive rules .
$M = 1$; indicates non linear additive rules.
P, Q, R, S and T can be either 0, meaning no connectivity, or 1 meaning connectivity. They are called the *impact coefficients* for cell i.
$\oplus$ denotes XOR
$\bullet$ denotes AND
where $P \bullet T \neq 1$.

For a 2 state 4NCA there are 2^5 distinct configurations and 2^{2^5} distinct mappings from all these neighborhood configuration to the next state.

2.2 Classification of 4NCA rules

In this section we define the naming convention for different types of connectivities each cell of a linear 4NCA can have with its neighboring cells. Denoting the state of cell i at time t by $q[i](t)$, the next state of cell i can depend on 4 cells (including any three neighboring cells and itself). Based on neighborhood dependency each CA cell can have one of the 4 classes of connectivity.

If the next state of cell i depends on only one of the 4 cells, the corresponding cell is said to have Class 1 connectivity. If next state depends on 2, 3 or 4 neighbors then the corresponding cell has Class 2, Class 3 or Class 4 connectivity respectively. A rule having Class i connectivity for each cell is called Class i rule, where $1 \leq i \leq 4$. For example 2863311530 is a Class 1 rule and 1019462460 is a Class 4 rule. There are 23 different connectivities each cell can have for our present model of the 4NCA. Therefore 23 different rules are applicable for a cell. Out of these, 5 rules belong to Class 1, 9 belong to Class 2, 7 belong to Class 3 and 2 belong to Class 4.

2.3 Entropy as a Measure of Randomness

The quality of random numbers can be measured in a variety of ways. One common method is to compute the information density, or entropy, in a series of numbers. The entropy of X is the uncertainty about the outcome before an observation of X. In other words entropy is a measure of the amount of unpredictable information there is in a data source. A sequence of good random numbers will have a high level of entropy.
Let k be the number of possible state values of each cell. The cell state sequences (bit strings) are divided into subsequences of length h, denoted by E_h to calculate the entropy of the cell's state sequence. So there are k^h possible subsequences of length h. Thus, entropy of bit string E_h can be given by:

$$E_h = -\Sigma_{j=1}^{k^h} p_{h_j} log_2 p_{h_j}.$$

where h_j is the j^{th} subsequence, $1 \leq j \leq k^h$ and p_{h_j} is the probability of subsequences h_j. The entropy E_h attains the maximum value when the probabilities p_{h_j} are same and equal to $1/k$. Thus, the randomness of the sequences of state of a cell can be judged according to the value of entropy. High entropy is a necessary, but by no means sufficient, condition for obtaining high-quality RNGs. In general, a battery of tests must be applied to this pattern to ascertain whether the evolved RNGs are indeed of high quality.
Once a high quality RNG is designed it is used to produce pseudorandom patterns in the following way:

1. The obtained n cell CA is initialized with a random seed.
2. It is then run for l time steps, to produce n random sequences of l bits.
3. These sequences are connected to form one long sequence of $n * l$ bits.
4. This process is repeated m times, thus a binary pattern of $m * n * l$ binary bits are obtained.

3 Probabilistic Analysis of 4NCA rules

The following notations hold for the present discussion on probabilistic analysis of 1-dimensional 4NCA rules having n cells with each cell having the same class of connectivity.
$q(t)$ represents the state of CA at time t.
$p(t)$ represents the frequency count of 1 in $q(t)$ i.e,

$$p(t) = \frac{\text{Numbers of cells having 1}}{\text{Total number of cells}}.$$

$p_i(t)$ denotes the probability of occurrence of 1 in i^{th} cell at time t.
$p_{i_{avg}}(t)$ can be deduced from $p_i(t)$ as follows:

$$p_{i_{avg}}(t) = \frac{\Sigma_{i=1}^{n} p_i(t)}{n}.$$

c_i denotes the count value for cell i.
$\delta(t)$ represents the deviation of $p(t)$ from 0.5 i.e,

$$\delta(t) = p(t) - 0.5.$$

$|\delta_{i_{avg}}(t)|$ represents the average of the modulus of the deviation of $p_i(t)$ from 0.5 at time t i.e,

$$|\delta_{i_{avg}}(t)| = \frac{\Sigma_{i=1}^{n} |p_i(t) - 0.5|}{n}.$$

In the sections to follow it is mathematically shown and experimentally verified that for the i^{th} cell $p_i(t+1)$ depends on $p(t)$ and the class of rule the cell is having.
Let us explain how $p_i(t+1)$ is obtained from $p(t)$ for a non complemented linear CA. We take the example of a Class 4 rule.
Assume that the cell i is connected to cells $i-1$, $i+1$, $i+2$ and itself. Since we are considering only linear CA, cell i will have a 1 in it at time $t+1$ if and only if out of cells $i-1$, i, $i+1$ and $i+2$:

1. Any one of the cells has a 1 in it at time t.
2. Any three of the cells have 1s in them at time t.

Mathematically,

$$\begin{aligned} p_i(t+1) &= 4(p(t)(1-p(t))^3 + (1-p(t))p(t)^3) \\ &= 4p(t)(1-p(t))(2p(t)^2 - 2p(t) + 1) \end{aligned}$$

Theoretically, for an n cell CA, $p_{i_{avg}}(t+1)$ is calculated as follows:

$$p_{i_{avg}}(t+1) = \frac{\Sigma_{i=1}^{n} p_i(t+1)}{n}.$$

For experimental purpose $p_{i_{avg}}(t+1)$ is calculated as follows:

```
1. For $i = 1, 2, \ldots$ n do
   (a) $c_i = 0$.
   (b) For a fixed value of $p(t)$ repeat $m$ times.
       i. Initialize CA with a random seed and evolve the CA for 1 time step. Increment $c_i$ if $q_i(t+1)$ is 1.
   (c) Set $c_i$ to $\frac{c_i}{m}$.
2. $p_{i_{avg}}(t+1) = \frac{\Sigma_{i=1}^{n} c_i}{n}$.
```

For an n cell CA, the above procedure is repeated $n+1$ times, each time with a different value of $p(t)$. Initially with $p(t)$=0 and incrementing $p(t)$ by $1/n$ at each iteration we get $n+1$ values of $p_{i_{avg}}$.

For our experiment we have n=100 and m=100. Table 1 gives the relationship between $p_{i_{avg}}(t+1)$ and $p(t)$ for an n cell periodic boundary, non complemented CA. Fig. 3 shows the ideal plot of $p_{i_{avg}}(t+1)$ against $p(t)$ with $p(t)$ on x-axis and $p(t+1)$ on y-axis. In order to experimentally verify the theoretical results, experiments were carried out to obtain various values of $p_{i_{avg}}(t+1)$ for different values of $p(t)$ for a CA of size n=100. Fig. 4 shows the plot of $p_{avg}(t+1)$ against $p(t)$ for the experiments conducted. As can be seen from Fig. 3 and Fig. 4, the experimentally obtained plot for the different classes of rules closely follow the theoretically obtained plots.

Table 1: $p_{i_{avg}}(t+1)$ represented in terms of $p(t)$

Class	$p_{i_{avg}}(t+1)$
Class 1	$p(t)$
Class 2	$2p(t)(1-p(t))$
Class 3	$p(t)(4p(t)^2-6p(t)+3)$
Class 4	$4p(t)(1-p(t))(2p(t)^2-2p(t)+1)$

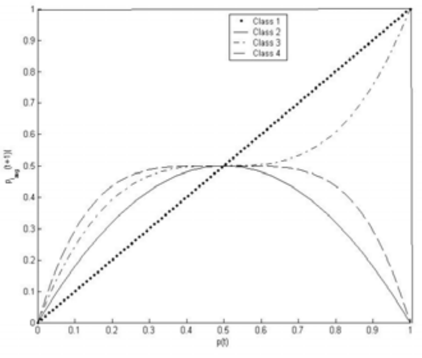

Figure 3: Theoretical plot of $p_{i_{avg}}(t+1)$ Vs $p(t)$

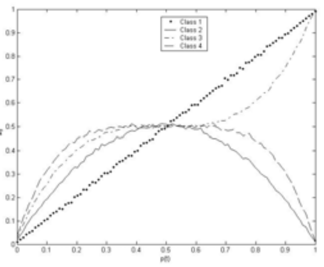

Figure 4: Experimental plot of $p_{i_{avg}}(t+1)$ Vs $p(t)$

Table 2 represents $|\delta_{i_{avg}}(t+1)|$ in terms of $\delta(t)$ for n cell periodic boundary, non complemented CA for various classes of rules. Fig. 5 shows the ideal plot of $|\delta_{i_{avg}}(t+1)|$ against $\delta(t)$ with $|\delta_{i_{avg}}(t+1)$ on y-axis and $\delta(t)$ on x-axis. In order to experimentally verify the theoretical results, experiments were carried out to obtain various values of $|\delta_{i_{avg}}(t+1)|$ for different values of $\delta(t)$ for a CA of size n=100. Fig. 6 shows the plot of $|\delta_{i_{avg}}(t+1)|$ against $\delta(t)$ for the experiments conducted. As earlier, the experimentally obtained plots closely follow the theoretical ones.

Table 2: $|\delta_{i_{avg}}(t+1)|$ expressed in terms of $\delta(t)$

Class	$\lvert\delta_{i_{avg}}(t+1)\rvert$
Class 1	$\lvert\delta(t)\rvert$
Class 2	$\lvert 2\delta(t)(1-\delta(t))\rvert$
Class 3	$\lvert\delta(t)(4\delta(t)^2-6\delta(t)+3)\rvert$
Class 4	$\lvert 4\delta(t)(1-\delta(t))(2\delta(t)^2-2\delta(t)+1)\rvert$

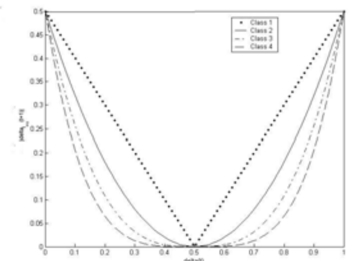

Figure 5: Theoritical plot of $|\delta_{i_{avg}}(t+1)|$ Vs $\delta(t)$

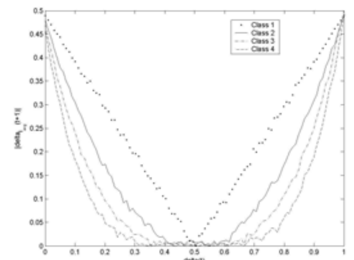

Figure 6: Experimental plot of $|\delta_{i_{avg}}(t+1)|$ Vs $\delta(t)$

From the above discussion it can be seen that $\delta_{i_{avg}}(t+1)$ is the minimum for any value of $\delta(t)$ for Class 4 rules followed by Class 3, 2 and 1 rules in that order. Hence substitution of Class 4 rules tends to equalize the probability of occurrence of 0 and 1 more than any other class of rule. This leads to maximization of entropy which is necessary (but not sufficient) criterion to ensure randomness. This forms the basis of the proposed rule selection algorithm discussed in the next section.

4 Proposed Rule Selection Algorithm (PRSA)

The proposed rule selection algorithm works as follows.

1. Select a Class 4 connectivity and substitute it to all the cells.

2. Select $m = (.1 * n)$ cells randomly. For each of the selected cells choose one of its impact coefficient (described in section 2.1) randomly and invert it, still producing a valid rule.

3. If resultant rule has entropy $\geq$ .998 * MAXIMUM ENTROPY, then stop the process else go to Step 1.

5 Results

The experiments were carried out for a 4NCA having 50 cells. The 50 cells were initialized with a random seed. The 4NCA was executed for 65536 time steps to produce 50 random sequences of 65536 bits. These 50 bit sequences are concatenated to form one long sequence of 3276800 bits. This process is repeated 30 times. Thus we obtain a sequence of little more then 98 million binary bits. This produces random data of 11.7 MB.

For evaluating the randomness of the generated bit string, we used what is probably the most stringent suite of randomness tests presented to date: Marsaglia's Diehard suite [5]. A detailed description of the tests is beyond the scope of this paper and the interested reader is referred to [5].

A comparison of the performance of the proposed technique is made with those of some common generators namely Shift Register Generator (SR), Extended Congruential Generator (EC) and Lagged Fibonacci Generator (LF). The results for these generators are taken from [15]. Another CA based RNG using an evolutionary technique is called cellular programming has been proposed in the literature [11]. The goal of the evolutionary algorithm is to evolve "good" rule tables for a nonuniform CA, i.e., rules that give rise to high-quality sequences of random numbers. For the purpose of comparison, the results for CP evolved CAs, CA1, CA2, CA3 are taken from [2]. Using the CA based approaches namely PRSA and CP, three different CAs are evolved thus providing three sets of results each. The diehard tests results are provided in Table 3. The results show that the bit patterns generated using PRSA evolved CA passes more number of diehard tests than any other competing method. PRSA is found to provide quite consistent test results for the three CAs, CA1, CA2 and CA3, while those obtained using CP provide varying performance. This indicates the effectiveness of the proposed RNG.

Comparing the timing requirements of the proposed and 50 cell CP based RNG, we found that while the former took 6 seconds, the latter required 12 minutes and 4 seconds to evolve the CAs on a 3GHz Intel Pentium 4 machine

Table 3: Test results. NA stands for Not Applied

Test Name	SR	EC	LF	CP			PRSA		
				CA1	CA2	CA3	CA1	CA2	CA3
Birthday Spacing	Y	Y	N	Y	Y	Y	Y	Y	Y
Tough Birthday Spacing	NA	NA	NA	N	Y	Y	Y	Y	Y
OPERMS Permutation 1	Y	Y	Y	Y	Y	Y	Y	Y	Y
OPERMS Permutation 2	Y	Y	Y	Y	Y	Y	Y	Y	Y
Binary Rank for 31X31 Matrices	N	Y	Y	Y	Y	Y	Y	Y	Y
Binary Rank for 32X32 Matrices	N	Y	Y	Y	Y	Y	Y	Y	Y
Binary Rank for 6X8 Matrices	Y	Y	Y	Y	Y	Y	Y	Y	Y
Bitstream Test	NA	NA	NA	N	N	N	N	N	Y
OPSO	NA	NA	NA	N	N	N	N	N	Y
OQSO	NA	NA	NA	N	N	N	N	N	Y
DNA	NA	NA	NA	N	N	N	N	N	Y
Count The 1's	Y	Y	N	Y	Y	Y	Y	Y	Y
Parking Lot	Y	Y	Y	Y	Y	Y	Y	Y	Y
Minimum Distance	Y	Y	N	N	N	N	N	N	Y
3D Sphere	Y	Y	Y	Y	Y	Y	Y	Y	Y
Squeeze	Y	Y	Y	Y	Y	Y	Y	Y	Y
Overlapping Sums	Y	Y	Y	Y	Y	Y	Y	Y	Y
Runs up 1	Y	Y	Y	Y	Y	Y	Y	Y	Y
Runs down 1	Y	Y	Y	Y	Y	Y	Y	Y	Y
Runs up 2	N	N	Y	Y	Y	Y	Y	Y	Y
Runs down 2	Y	Y	Y	Y	Y	Y	Y	Y	Y
Craps Test1	Y	Y	N	Y	Y	Y	Y	Y	Y
Craps Test2	Y	Y	N	Y	Y	Y	Y	Y	Y

Table 4: Comparison between average CPU time taken by PRSA and CP to evolve CA rules

	PRSA	CP
Average CPU Time	6(sec)	12(min) 4(sec)

6 Conclusion

Based on a probabilistic analysis, we construct a random number generator using a one dimensional, non-uniform 4NCA. The analysis points to the fact that Class 4 rules are likely to provide better random patterns than Class 3, 2 or 1 rules. The random patterns generated using the proposed method are compared with those generated using a CP based method and several other existing methods. The results show that the present approach outperforms CP both in terms of average time taken to evolve CA rules and in terms of quality of pseudo random patterns generated. The generator based on 4NCA can produce a high quality of random patterns which pass most of, sometimes all, the tests. The 4NCA generator can produce random numbers quickly and can be implemented conveniently by hardware, and can be used in many fields such as built-in-self-test of VLSI and Cryptography. The authors are working in these directions.

References

[1] G. P. Biswas. Modification of 3-neighborhood cellular automata to 4-neighborhood cellular automata to increase their capabilities. *2007Journal of CSI*, 33:27–34, 2003.

[2] G. P. Biswas, R. K. Mishra, Abhishek Seth, Prashant Golash, and Ranjan Pandey. Design of 4-neighborhood cellular automata based high quality random pattern generator using genetic algorithm. *Proceedings of the National seminar of recent advances on Information Technology (RAIT-2007)*, 2007.

[3] K. Cattel, S. Zhang, M. Serra, and J.C. Muzio. 2-by-n hybrid cellular automata with regular configuration: Theory and application. *IEEE Trans. Computers*, 48(3):285–295, March 1999.

[4] D. R. Chowdhury, S. Nandy, and S. Chattopadhyay. Additive Cellular Automata: Theory and Applications. *Journal*, 1(2):12–15, January 1994.

[5] G. Marsaglia. Diehard battery of tests.

[6] S. Nandi, B. K. Kar, and P.P.Chaudhuri. Theory and application of cellular automata in cryptogrphy. *IEEE Trans. Computers*, 43:1,346–1,357, 1994.

[7] S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Communications of the ACM*, 31(10):1192–1201, October 1988.

[8] Marcin Seredynski, Krzysztof Pienkosz, and Pascal Bouvry. Reversible cellular automata based encryption. In Hai Jin, Guang R. Gao, Zhiwei Xu, and Hao Chen, editors, *NPC*, volume 3222 of *Lecture Notes in Computer Science*, pages 411–418. Springer, 2004.

[9] M. Sipper. The emergence of cellular computing. *Computers*, 32:7, July 1999.

[10] M. Sipper and M. Tomassini. Generating parallel random number generators by cellular programming. *Modern Physics C*, 7(2):181–190, 1996.

[11] M. Sipper and M. Tomassini. Computation artificially evolved, non-uniform cellular automata. *Theoretical Computer Science*, 217:81–98, 1999.

[12] M. Tomassini, M. Sipper, M. Zolla, and M. Perrenoud. Generating high-quality random numbers in parallel by cellular automata. *Future Generation Computer Systems*, 16:291–305, 1999.

[13] S. Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55:601–644, 1983.

[14] S. Wolfram. Random sequence generation by cellular automata. *Advances in Applied Mathematics*, 7:123–169, 1986.

[15] Z.Xuelong, LQianmu, X.Manwu, and L.Fengyu. A symmetric cryptography based on extended cellular automata. *IEEE International Conference on Systems, Man and Cybernetics*, 1:499–503, 2005.