Generating High-Quality Random Numbers by Next Nearest-Neighbor Cellular Automata

Ping Ping[a], Feng Xu[b] and Zhi-Jian Wang[c]

College of Computer and Information, Hohai University, Nanjing 210098, China

[a]pingpingnjust@163.com, [b]xufeng@hhu.edu.cn, [c]w51178@sina.com

Keywords: cryptography; random number generator; cellular automata; NIST statistical test suite

Abstract. Cellular automaton (CA) has been widely investigated as random number generators (RNGs). However, the CA rule and the number of neighbors must be chosen carefully for good randomness. In Ref. [11], non-uniform CA with next nearest neighborhood was applied to generate a pseudo-random sequence. Considering that non-uniform CA has more complex implementation in hardware and needs lager memory to store different rules than uniform CA. In this paper, we propose new RNGs based on uniform CA with next nearest neighborhood. Time spacing technique and NIST statistical test suite are used to find optimal rules for uniform CA. Experiment results show that the sequences generated by uniform CA with optimal rules successfully passed all tests in the NIST suite.

Introduction

Random number generators (RNGs) are useful for many purpose including cryptography devices, Monte Carlo simulations, Built-In Self-Test circuits, etc. In the last decades, cellular automaton (CA)-based RNGs were studied extensively [1-4]. The notable features of CA, such as simple components, regular structure, local interconnection and massive parallelism, make them easier and faster to implement in hardware than other models.

Wolfram [5] was the first researcher who proposed a random number generator by one-dimensional (1-D) uniform CA with rule 30. But it was later cryptanalyzed by Miere and Stafflebach [6] mainly due to its correlation. Subsequently, many researchers focused on non-uniform CA [7-9], where each cell may contain a different rule in contrast to the uniform CA. In [10], an evolutionary technique called cellular programming was proposed and a set of 47 rules with radius one and two was discovered by using cellular programming. These rules, characterized with high values of entropy, were potentially suitable for generating high-quality random sequences. It was shown that the performance of non-uniform CA based RNGs are superior to that of uniform CA. In [11], the above set is reduced to five rules with radius two by a genetic algorithm and experiments was shown that each set of rules passed all FIPS 140-2 tests. In these non-uniform CA based works, rules for non-uniform CA play an important role in the quality of the random sequence, so they must be chosen carefully. The main concern of this paper is to design random number generators based on uniform CA with next nearest neighborhood. To find appropriate rules for uniform CA, time spacing technique and NIST statistical test suite are used. Experimental results show that the suggested random number generators have good randomness.

Cellular Automaton

Cellular automata are abstract dynamical systems in which state, space and time are discrete. A 1-D cellular automaton is defined as 1-D lattice of cells, each of which can take a finite number of discrete states, updated synchronously in discrete time steps, according to a local, identical, deterministic rule:

$$s_i^{t+1} = f(s_{i-r}^t, ..., s_i^t, ..., s_{i+r}^t), \quad (1)$$

Here, s_i^t denotes the state of cell i at time step t, r is the rule radius, and the local rule f is a combinatorial function that gives the new state of a cell in terms of the current states of all cells in its neighborhood. The nearest neighborhood, having a radius $r=1$, consists of three cells $s_{i-1}^t, s_i^t, s_{i+1}^t$. The next nearest neighborhood, having a radius $r=2$, consists of five cells $s_{i-2}^t, s_{i-1}^t, s_i^t, s_{i+1}^t, s_{i+2}^t$. In this work, we concentrate on the next nearest-neighbor CA which has two possible states (0 or 1), and rules with radius two. There are 2^{2^5} possible different rules for next nearest-neighbor CA. Each rule specified by a decimal number is conventionally referred to as rule number. When considering finite CA, cyclic boundary conditions are frequently applied. If all CA cells obey the same rule, then the CA is said to be uniform CA. Otherwise, it is called non- uniform CA.

Overview of Two Existing Algorithm

PNSs Based on Non-Uniform CAs with Rule Radius One and Two

In [10], CAs are applied to generate a pseudo-random sequence (PNS) which is used during the encryption process. Rules of radius $r=1$and 2 for non-uniform 1-D CAs have been considered. Instead of design rules for CAs, an evolutionary technique called cellular programming (CP) is employed to discover rules for non-uniform CAs. In the result of CP searching process a set of 47 rules, including 8 rules of radius one and 39 rules of radius two, was found. Then, these rules which are characterized with high values of entropy are further tested by FIPS 140-2 and Marsaglia statistical tests. As a result, a small set of 8 rules has been selected: 30, 86, 101 ($r=1$), and 869020563, 1047380370, 1436194405, 1436965290, 1705400746, 1815843780, 2084275140, 2592765285 ($r=2$). Experimental results show that selected rules working collectively in non-uniform CA are able to produce high quality PNSs. However, works in [16] point out that some specific assignments of these rules to CA cells may lead to poor quality of PNSs.

PNSs Based on Non-Uniform CAs with Rule Radius Two

Considering that the quality of PNSs highly depends on the assignments of applied rules, a genetic algorithm (GA) is used in [11] to find suitable rules from the set of 47 rules. Some sets of bad combination of rules are eliminated by GA and 10 subsets of rules are finally selected from 47 rules. All these 10 subsets are composed of rules from the set of five rules: 1436194405, 1436965290, 1721325161, 1704302169, 1705400746 ($r=2$). The advantage of this method is that high quality of PNSs can be obtained by using non-uniform CAs with any assignments of rules in the sets.

Finding Optimal Rules of Radius $r=2$ by NIST Test Suite

Many researchers focused on non-uniform CA for cryptographic application. But, non-uniform CAs have more complex hardware implementation and need lager memory to store different rules than uniform CAs. The purpose of this paper is to find uniform CA rules of radius $r=2$ that the PNSs produced by them have good random property.

NIST Statistical Test Suite

Among the numerous standard tests for randomness, a convincing way to show the randomness of the produced sequences is to confront them to the NIST (National Institute of Standards and Technology) Statistical Tests [12]. The NIST STS (statistical test suite) consisting of 15 tests. These tests focus on variety of different types of non-randomness that could exist in a sequence.

NIST suggests two approaches to interpret test results: the proportion of sequences passing a test and the distribution of P-values. A significance level $\alpha=0.01$, as recommended by NIST, was used for the analysis of P-values obtained from various tests. A sequence passes a statistical test whenever the P-value$\geq \alpha$ and fails otherwise.

(1) Proportion of sequences passing a test

The range of acceptable proportions is defined as,

$$\hat{p} \pm 3\sqrt{\frac{\hat{p}(1-\hat{p})}{m}}, \tag{2}$$

where $\hat{p} = 1 - \alpha$, and m is the sample size. If the proportion falls outside of this interval, then there is evidence that the data is non-random. For instance, if m=100, so the range of acceptable proportion is 0.99 ± 0.02985 ($[0.96015, 1.01985]$).

(2) Distribution of P-values

The distribution of P-values is checked to ensure uniformity. The interval [0, 1] is divided into 10 sub-intervals and the computation is as follows:

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - m/10)^2}{m/10}, \tag{3}$$

where F_i is the number of occurrences that the P-values is in sub-interval i and m denotes the sample size. A P-value is calculated such that

$$P - value_T = igamc(\frac{9}{2}, \frac{\chi^2}{2}), \tag{4}$$

where igamc is the incomplete Gamma function. If $P - value_T \geq 0.0001$, then the sequences can be considered to be uniformly distributed.

In our following tests, parameters used for NIST test suite are listed in Table 1.

Table 1 Parameters used for NIST Test Suite

Statistical Test Name	Test Parameter	Statistical Test Name	Test Parameter
Block Frequency	M=20000	Linear Complexity	*M*=500
Non-overlapping Templates	*m*=9, *B*=000000001	Cumulative Sums	Forward
Overlapping Templates	*m*=9, *B*=111111111	Random Excursions	*x*=-1
Approximate Entropy	*m*=10	Random Excursions Variant	*x*=-1
Serial	m =16, $\nabla\Psi_m^2$		

Finding Optimal Rules

Instead of finding appropriate rules in huge rule space, we use a new set of 10 rules of radius $r = 2$: R1=1436194405, R2=1436965290, R3=1721325161, R4=1704302169, R5=1705400746, R6=869020563, R7=1047380370, R8=1815843780, R9=2084275140, R10=2592765258, which has been discovered by [10] and [11]. All these rules are tested by NIST statistical test suite. The method of testing is given as follows:

Step 1: One rule is selected from the new set.

Step 2: Uniform CA consisting of 50 cells is initialized with 50 bit random stream. Periodic boundary is adopted.

Step 3: This CA evolves 80000 time steps with the selected rule and generates 4000000-bit sequence.

Step 4: Time spacing technology [13] is applied to decorrelate bit sequence. Here, a time-spacing parameter of 4 is adopted, which means only bits of time steps 0, 4, 8,… are considered as part of the output sequence. Thus, we obtain a 1000000-bit output sequence.

Step 5: Repeat Step 2 to step 4 100 times so as to get 100 samples of 1000000-bit output sequence.

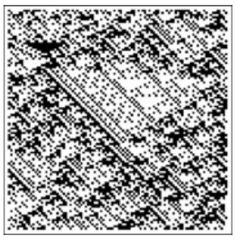
Step 6: 100 samples of different bit sequence are tested by NIST STS (version 2.1.1).

The results of proportions of the sequences passing the tests for 9 rules are presented in Table 2. The test result of rule 2592765258 has not been included in Table 2, because it even cannot pass Frequency Test which is the basis of all subsequent tests. In fact, the space-time diagram for rule 2592765258 tells us why it cannot pass the Frequency Test. Fig. 1 shows a space-time diagram for rule 2592765258. The black block represents '1' and the white block represents '0'. It is well known that the number of ones and zeroes in a sequence should be about the same for a truly random sequence. But, it is obvious that the number of zeroes is more than the number of ones in Fig.1. So the sequence generated by rule 2592765258 is not random enough.

From Table 2, it can be seen that rule 1047380370 fails to pass four tests. Rules 1436194405, 869020563, 1815843780 and 2084275140 fail to pass two tests. Rule 1704302169 gives better results but still has one failed tests. The highest randomness is obtained using rules 1436965290, 1721325161, 1705400746 which have passed all tests.

If the test sequences are truly random, P-value is expected to appear uniform in [0, 1]. Hence, the randomness of rule 1436965290, 1721325161, 1705400746 is further checked by uniformity of P-value of the test samples. Fig. 2 illustrates the distribution of P-values for three rules. Dashed line represents the threshold value of $P-value_T$. It is easy to see that all $P-value_T$ are above the threshold 0.0001, which means the sequences produced by three tested rules can be considered to be uniformly distributed.

Through the above analysis, the optimal rules for uniform CA generating high quality PNSs have been found. There are rules 1436965290, 1721325161, 1705400746, which passed all NIST statistical tests.


(a) No time spacing

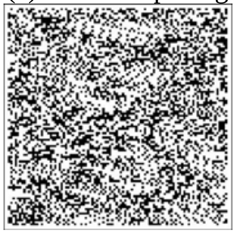

(b) Time spacing of 4

Fig. 1 Space-time diagram for rule 2592765258

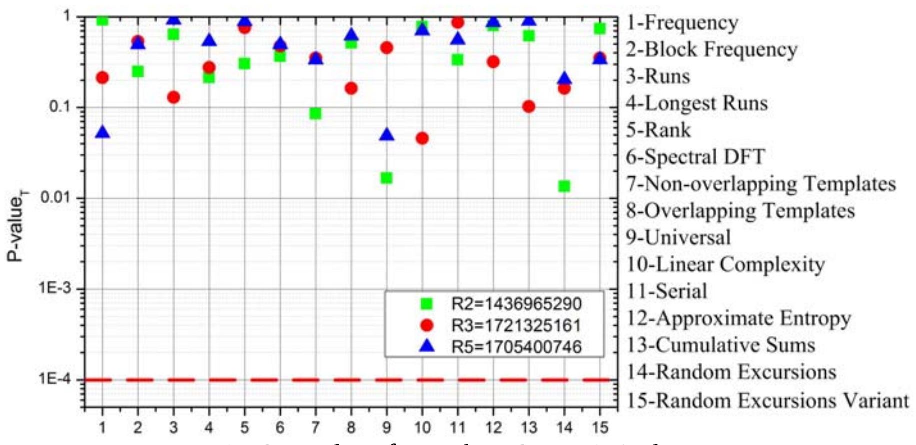


Fig. 2 P-value$_T$ for each NIST statistical test

Table 2 The results of proportion`s of the sequences passed the tests

No.	Statistical Test	Proportion								
		R1	R2	R3	R4	R5	R6	R7	R8	R9
1	Frequency	1.00	0.99	1.00	0.99	0.98	1.00	0.97	1.00	0.96
2	Block Frequency	0.94*	0.99	1.00	0.98	0.98	1.00	0.87*	0.98	0.97
3	Runs	1.00	0.97	1.00	0.98	0.99	1.00	1.00	0.98	0.98
4	Longest Runs	0.99	0.99	0.97	0.98	0.98	1.00	0.99	0.99	0.97
5	Rank	1.00	1.00	1.00	1.00	1.00	0.98	1.00	0.97	0.99
6	Spectral DFT	0.98	0.99	0.98	0.97	0.98	0.00*	0.07*	0.98	0.97
7	Non-overlapping Templates	0.99	0.98	1.00	0.98	1.00	0.99	0.98	1.00	0.98
8	Overlapping Templates	0.99	0.99	0.99	0.99	0.99	0.96	0.99	1.00	0.98
9	Universal	0.13*	0.98	0.97	0.23*	0.96	0.94*	0.99	0.98	0.98
10	Linear Complexity	1.00	1.00	0.98	0.98	1.00	1.00	1.00	0.99	1.00
11	Serial	1.00	0.99	0.99	0.98	0.99	0.99	0.00*	0.70*	0.43*
12	Approximate Entropy	1.00	0.97	0.99	0.98	0.98	0.99	0.00*	0.33*	0.04*
13	Cumulative Sums	1.00	0.98	1.00	0.99	0.97	1.00	0.97	0.99	0.97
14	Random Excursions	1.00	1.00	1.00	0.98	0.98	1.00	0.98	0.97	1.00
15	Random Excursions Variant	1.00	1.00	0.98	1.00	1.00	0.96	0.98	1.00	0.98

Conclusions

In this paper we concentrate on generating pseudo-random sequences by employing uniform CA with next nearest neighborhood. To find appropriate rules, NIST statistical test suite and a sampling technique called time spacing technique are used. As a result, three optimal rules of radius $r = 2$ (1436965290, 1721325161, 1705400746) are discovered from a set of 10 rules. Experiment results demonstrate that uniform CA based RNGs can perform at least as well as non-uniform CA based RNGs if appropriate rules are applied. With respect to hardware implementation and memory requirement, uniform CA outperform non-uniform CA for its simple structure and identical rule.

References

[1] P. D. Hortensius, R. D. McLeod, W. Pries, D. M. Miller and H. C. Card, Cellular automata-based pseudorandom number generators for Built-In Self-Test, IEEE Trans. Comput. Aid. D 8 (1989) 842-859.

[2] S. U. Guan, S. Zhang, An evolutionary approach to the design of controllable cellular automata structure for random number generation, IEEE Trans. Evolut. Comput. 7 (2003) 23-36.

[3] S. U. Guan, S. K. Tan, Pseudorandom number generation with self-programmable cellular automata, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 23 (2004) 1095–1101.

[4] B. H. Kang, D. H. Lee, C. P. Hong, High performance pseudorandom number generator using two-dimensional cellular automata, In: 4th IEEE International Symposium on Electronic Design, Test & Application, Enero, 2008, pp. 597-602.

[5] S. Wolfram, Random sequence generation by cellular automata, Advances in applied mathematics 7 (1986) 123-169.

[6] W. Meier, O. Staffelbach, Analysis of pseudo random sequences generated by cellular automata, Advances in Cryptology: Proceedings of EUROCRYPT, 1991, pp. 186-199.

[7] M. Tomassini, M. Sipper, and M. Perrenoud, On the generation of high-quality random numbers by two-dimensional cellular automata, IEEE Trans. Comput. 49 (2000) 1146-1151.

[8] R. Dogaru, Hybrid cellular automata as pseudo-random number generators with binary synchronization property, International Symposium on Signals, Circuits and Systems, 2009, pp. 1-4.

[9] K. Chakraborty, D. Chowdhury, CSHR: Selection of Cryptographically Suitable Hybrid Cellular Automata Rule, In: 10th International Conference on Cellular Automata for Research and Industry, ACRI 2012, pp. 591-600.

[10] F. Seredynski, P. Bouvry, A. Zomaya, Cellular automata computation and secret key cryptography, Parallel Computation 30 (2004) 753-766.

[11] M. Szaban, F. Seredynski, P. Bouvry, Collective behavior of rules for cellular automata-based stream ciphers, In: IEEE Congress in Evolutionary Computation, 2006, pp. 179-183.

[12] Rukhin A et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. NIST Special Publication 800-22 (with revisions dated April, 2010).

[13] M. Tomassini, M. Sipper, M. Zolla, M. Perrenoud, Generating high-quality random numbers in parallel by cellular automata, Future Generation Computer System 16 (1999) 291-305