SAC: G: 3-D Cellular Automata based PRNG

Rosemary Koikara
Kungpook National University
School of Computer Science and Engineering
Daegu, South Korea
rosekoikara@gmail.com

ABSTRACT

Random numbers are critical in many areas like security, simulation, gaming and even gambling. Random numbers are basically of two types: true random numbers and pseudo-random numbers. In this paper we propose a three dimensional cellular automata (3-D CA) based pseudo-random number generator (PRNG). Cellular Automata (CA) is used in pseudo-random number generators to produce high-rate random numbers. However, the randomness of such numbers directly depends upon the CA rules and the number of neighbor cells. The two-dimensional (2-D) CA have several limitations such as finding the best CA rules, boundary cell problems, etc. To address the problems existing in the 2-D CA, we propose a random number generator based on 3-D cellular automata. The proposed generator is based on the rule numbers 43, 85, 170 and 201, and on incremental boundary conditions. The output bits are then passed to the well-known Diehard, ENT and NIST test suites to test its randomness. The results reveal that the bit stream generated by the proposed scheme passes all the tests present in the test suites.

Keywords

Pseudo-random number generator; Cellular Automata; Security

1. PROBLEM AND MOTIVATION

Pseudo-random numbers are required in any application that requires unpredictable results. In such systems the quality of these systems hugely depend on the quality of the random numbers used. Since the sequence of random numbers that have to be produced are generally very long, computational efficiency is also an important aspect. The goal is to obtain pseudo random numbers that are like true random numbers. Several algorithms have been proposed throughout the years to produce pseudo-random numbers, with each algorithm stronger and more efficient than the other. Some of the most common deterministic algorithms used were linear congruential generator [12], Linear Feedback Shift Register [11], and lagged Fibonacci generators [11]. Many statistical tests have been developed to test randomness of the sequences of numbers generated by PRNGs [22]. These schemes show poor performance against Diehard, ENT and NIST test suites. Wolfram [24] first proposed the use of one-dimensional cellular automata for the generation of random sequences of bits. A cellular automaton can be defined as a collection of cells arranged in such a way that each cell changes its state according to the change in time and depending on the states of their neighbors. The change in the cells depends on an update mechanism and various rules that affect the updation. The change in the cell happens simultaneously and with the same update rule. A cellular automaton may be one-dimensional, two-dimensional or three-dimensional. The motivation for the use of cellular automaton in random number generation comes from the fact that cellular automaton are capable of complex behavior and they enable massive parallel computation [25]. However, the 1-D CA is no longer statistically strong against security attacks. Therefore, researchers proposed to use the two-dimensional cellular automata (2-D CA). Chowdhury et. al [5] proposed the use of two-dimensional cellular automata to enable the generation of high quality pseudo-random numbers. The randomness of these numbers directly depend upon the CA rules and the neighbor cells. The 2-D CA along with the unique boundary conditions and efficient rules produce more complex random numbers compared to the 1-D CA [22]. The 2-D CA has several limitations such as, finding the best CA rules, boundary cell problems, etc. Therefore, to address these challenges and problems in the existing PRNGs, we propose a 3-D CA based on the incremental boundary conditions and rule numbers 43, 85, 170 and 201. The proposed scheme passes all the Diehard, ENT and NIST test suites.

http://dx.doi.org/10.1145/0000000.0000000

2. BACKGROUND AND RELATED WORK

To date several deterministic algorithms have been proposed for producing random numbers. Deterministic algorithms are a more practical approach to generating random numbers. Hence they are called pseudo-random numbers. They can be differentiated from true random numbers, which are random numbers produced as a result of some natural physical phenomenon. The application of random numbers is

extensively used in different fields such as Biocomputing, engineering, security, Monte-Carlo simulations, etc. Traditionally, random numbers were generated by simple mathematical and statistical methods. However, with the passage of time researchers found that the random numbers generated by such systems have poor statistical properties. Therefore, researchers diverted their focus to complex mathematical methods. A random number should have certain properties in-order to be in elaborate stochastic simulations: (a) Long-period (b) Good statistical properties (c) Large linear complexity. Therefore, a random number generator that is computationally efficient must be designed to generate a random number that fulfills these properties. Most of the PRNGs are based on the Linear Feedback Shift Register (LFSR), linear congruential generator, etc.

Linear congruential generators [12] are one of the oldest PRNGs. It is a pseudo-random number generator that produces a sequence of numbers $r_1, r_2, r_3, \cdots$ of the form

$$r_t = ar_{t-1} + b(\mathrm{mod} m) \tag{1}$$

The above PRNG has a seed value r_0 and $t \geq 1$. In Equation 1 the integers a, b and m characterize the PRNG that will produce numbers between 0 and $m-1$. The value $0 \leq a \leq m$ is called the multiplier and the value $0 \leq a \leq b$ is called the increment. This generator passes the following statistical tests: Golomb's randomness postulates, frequency test, serial test, poker test, runs test, autocorrelation test and Maurer's universal statistical test. However, this sequence is predictable [15]. On observing the generator one can clearly see that after a maximum possible period of m, the sequence starts to repeat itself. Linear congruential generators were once very popular among researchers and most mathematical software packages.

Then there is the linear feedback shift register (LFSR) generator [[11]] which is also a commonly used PRNG. The LFSR is a generalized scheme of the LCG and is of the form

$$r_t = a_1 r_{t-1} + a_1 r_{t-1} + \ + a_m r_{t-m} (\mathrm{mod} p) \tag{2}$$

The LFSR generator has seed value r_0. In Equation 3 p is a prime number and determines the period of the generator that is $p^n - 1$.

In order to generate strong pseudo-random numbers, it is suitable for the generator to pass all standard empirical random number generator tests. But LCG fails at this. Hence, lagged Fibonacci generators (LFG) were developed as an improvement over linear congruential generators. The numbers generated are of the form

$$r_t = r_{i-j} \bigcirc r_{i-k} (\mathrm{mod} m), 0 < j < k \tag{3}$$

where $\bigcirc$ represents a binary operator, addition, multiplication or exclusive OR (XOR). Typically $m = 2^M$ with M=32 or 64. The seed is a block of size k. Depending on the operation used, i.e., addition, multiplication or XOR, it is either called an additive Fibonacci generator, a multiplicative lagged Fibonacci generator, or a two-tap generalized feedback shift register (GFSR) respectively.

Other commonly-used PRNGs are Mersenne Twister [13], WELL (Well Equidistributed Long-period Linear) generators [14], ran2 [16], drand48 and others. However, these schemes show poor performance against the Diehard and ENT test suites and certain schemes fail other tests too. Eventually, researchers proposed random number generators based on the Cellular Automata (CA).

Cellular automata is the fruit of the efforts of mathematician John von Neumann to create a self-replicating machine [18]. A cellular automaton is an accumulation of cells arranged in a grid in such a way that each cell changes state as a function of time. The changes are governed by a set of rules that incorporate the states of neighboring cells. Examples of cell shapes are squares, hexagons and cubes. The topology of cellular automaton is very compact. Elementary cellular automata, which is the simplest class of one-dimensional cellular automata, has two possible values of each cell (0 or 1). Here, the cells are on a straight line. The most popular example of cellular automaton is the "Game of Life" introduced by Conway in the 70's [1]. The reason for cellular automata being applicable is that they are mathematical systems constructed from many identical components, each simple, but together capable of complex behavior [25]. Also they enable massive parallel computation which is very desirable in various scenarios especially security. In theory, a CA can have any number of dimensions, and each cell can have any discrete steps at regular time intervals. The state of a cell at any given time depends on two things: (1) its previous state (2) the previous states of its immediate neighbors. The neighborhood in one-dimensional CAs comprises of r nodes on either side of the cell. For two-dimensional CAs the neighborhood either consists of 5 cells that include the cell itself and four immediate non-diagonal neighbors, and 9 cells that include the cell itself along with its eight surrounding neighbors. Also, there are various rules and update mechanisms that govern the next state of the cells. All the cells change their configuration simultaneously and using the same update rule. The update rule determines the values used in the update mechanism.

CA based PRNG shows better performance over the LFSR and other PRNGs. Wolfram [24] first suggested the use of one-dimensional, two-state CAs for the generation of random bit sequences. Cells are arranged in a line with each cell having the value 0 or 1. The updation rule for this automata is

$$a_i' = a_{i-1} \text{ XOR } (a_i \text{ OR } a_{i+1}) \tag{4}$$

After Wolfram introduced this, the one-dimensional CA has been extensively studied [4,8,9,21]. These are superior compared to the random number generators discussed previously [3]. However, the 1D CA is no longer statistically strong against security attacks. Chowdhury et al [5] proposed the use of 2-D CA for generating pseudo-random numbers. Comparison of the patterns generated by 2-D CA with those generated by LFSR and 1-D CA [10] prove that 2-D CA exhibit higher randomness. The 2D CA along with its unique boundary conditions and efficient rules produce more complex random numbers than the 2D CA. The 2-D CA as a pseudo-random number generator has been hugely researched [6, 7, 19, 20]. However, finding optimal rules in the case of 2D CA is still a challenging task [7]. Moreover, boundary conditions and application of a static rule structure still exists in the current CA-based PRNGs.

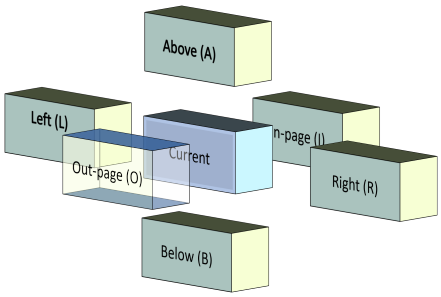

Figure 1: The neighborhood of a cell.

To address these challenges in the existing PRNGs, we propose a 3D CA based on the incremental boundary conditions and rule number 43, 86, 170 and 201. The proposed scheme passes all the tests present in the Diehard, ENT and NIST test suites, respectively.

3. 3-D CA BASED PSEUDO-RANDOM NUMBER GENERATOR

In this section a 3-D CA based PRNG is proposed. The motivation for using 3-D cellular automata for generating pseudo-random numbers was the fact that it generates high-quality random numbers and that the use of cellular automata encourages parallel computations and ease of hardware implementation. The cells are arranged in the form of a cube. Unlike one-dimensional and two-dimensional memory cellular automata, the three-dimensional memory cellular automata are distinct dynamic systems that comprise of finite three-dimensional block of cells. Each block consists of $n = d^3$ cells, where d is the length of each dimension. In this scheme the value of d is considered to be 512 bits. Each of the cells in the cellular automata holds a bit of value either 0 or 1. To increase the randomness of the numbers generated the initial configuration (the seed) of the cells are random. This PRNG also has two control bits.

Each cell at time t is represented as $a^{(t)}_{x,y,z}$. With time the CA gets evolved depending on the updating mechanism, control bits and the boundary conditions. The previous states of the neighboring cells influence the new state of each of the cells. At the end of each cycle a sequence 512 random bits are generated. In order to enhance the security, the bits at odd locations are interchanged with each other. For example, the first bit that is generated is interchanged with the third bit.

3.1 Neighborhood

Each 3-D cell is surrounded by six different cells called neighbors of the current cell. Hence, the neighborhood of cells that affect the state of a cell are in the left (L), right (R), above (A), below (B), in-page (I) and out-page (O) directions. Figure 3.1 shows the neighborhood of cells that affect the state of the current cell.

3.2 Boundary Conditions

As the states of each cell depends on the previous states of neighboring cells, the boundary conditions have to be considered. The purpose of the boundary conditions is to maintain the dynamics of the CA, such that operations for the state of the cells on the edge of the CA are well defined. The most commonly used boundary condition is the incremental boundary condition. In the case of incremental boundary conditions, the neighbor for cells residing at the boundary of the automata is randomly chosen from the rest of the cells. For example, the cells on the left side of the first cells and on the right side of the last cells are concatenated with a cell randomly selected from the rest of the cells. This leads to an increase in the complexity of choosing the cells. Therefore, in the proposed PRNG we modify the structure of the incremental boundary condition by selecting the next cells present on each side. In other words, the last cell in a row is the cell on the left side of the first cell, conversely, the first cell in a row is the cell on the right side of the last cell in the same row. Also, the last cell in a column is above the first cell in that column, conversely, the first cell in a column is the cell below the last cell of the same column. This ensures that all the cells in the 3D-CA are updated efficiently.

3.3 Updating Mechanism

As discussed earlier, the total number of possible configurations for the 3-D CA is 2^n, where $n = d^3$. Let, $\mathcal{C}$ be the set of all possible configurations of the 3-D CA, then $|\mathcal{C}| = 2^n$. According to time the state of the cells in 3D-CA changes, i.e., there is a change in configurations. This change is a transformation, $\mathcal{T} : \mathcal{C} \rightarrow \mathcal{C}$, i.e., at every discrete time interval, $C^{(t+1)} = \mathcal{T}(C^{(t)})$ where $(C^{(t)})$ is the configuration of the CA at time t. The 3-D cellular automata in this PRNG consists of 512 cells as each dimension consists of eight cells. Thus, in one cycle the proposed 3-D PRNG generates 512. If all the neighboring cells participate in the updating process, then the complexity of the system exponentially increases. Therefore, we use three different updating mechanisms (M). Whenever a cell needs to be updated, the proposed scheme randomly chooses one of the following three mechanisms.

$$\begin{aligned}
M^{(t)}_{x,y} : a^{(t)}_{x,y,i} &= C_1 \oplus C_2 \oplus (L \cdot a_{x-1,y,i}) \oplus (R \cdot a_{x+1,y,i}) \\
&\oplus (A \cdot a_{x,y-1,i}) \oplus (B \cdot a_{x,y+1,i}), 1 \leq i \leq 8 \\
M^{(t)}_{y,z} : a^{(t)}_{i,y,z} &= C_1 \oplus C_2 \oplus (A \cdot a_{i,y-1,z}) \oplus (B \cdot a_{i,y+1,z}) \\
&\oplus (I \cdot a_{i,y,z+1}) \oplus (O \cdot a_{i,y,z-1}), 1 \leq i \leq 8 \\
M^{(t)}_{x,z} : a^{(t)}_{x,i,z} &= C_1 \oplus C_2 \oplus (L \cdot a_{x-1,i,z}) \oplus (R \cdot a_{x+1,i,z}) \\
&\oplus (I \cdot a_{x,i,z+1}) \oplus (O \cdot a_{x,i,z-1}), 1 \leq i \leq 8
\end{aligned} \tag{5}$$

In Equation 5 C_1 and C_2 are clock bits and L, R, A, B, I and O represent the neighboring bits in the left, right, above, bottom, in-page and down-page directions. The symbols $\oplus$ and $\cdot$ represent the Boolean XOR and AND operators respectively. The proposed 3-D cell structure is shown in Figure 3.3. As the structure is three dimensional, the third axis is not visible in the figure. In one cycle, each cell in the 3D structure passes through rule number 43, 85, 170 and 201. These rules are shown in Table 1.

3.4 Algorithm

Input: An 3-D CA with dimensions $8 \times 8 \times \times 8$, i.e., 512 cells containing random bits of 0 or 1.

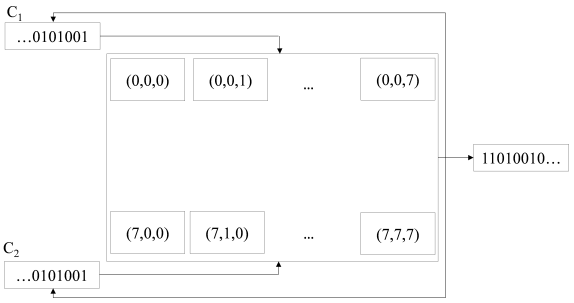

Figure 2: The 3-D cell structure of the proposed PRNG.

Table 1: Rules in the proposed PRNG

Rule No.	C_1	C_2	L	R	U	D	I	O
43	0	0	1	0	1	0	1	1
85	0	1	0	1	0	1	0	1
170	1	0	1	0	1	0	1	0
201	1	1	0	0	1	0	0	1

Output: $k \times 512$ random bits of 0 or 1.

Step 1: Randomly choose update mechanism $M_{x,y}^{(t)}$ or $M_{y,z}^{(t)}$ or $M_{x,z}^{(t)}$ given in Equation 5 and perform using rule numbers 43.

Step 2: Randomly choose update mechanism $M_{x,y}^{(t)}$ or $M_{y,z}^{(t)}$ or $M_{x,z}^{(t)}$ given in Equation 5 and perform using rule numbers 85.

Step 3: Randomly choose update mechanism $M_{x,y}^{(t)}$ or $M_{y,z}^{(t)}$ or $M_{x,z}^{(t)}$ given in Equation 5 and perform using rule numbers 170.

Step 4: Randomly choose update mechanism $M_{x,y}^{(t)}$ or $M_{y,z}^{(t)}$ or $M_{x,z}^{(t)}$ given in Equation 5 and perform using rule numbers 201.

Step 5: Interchange the odd bits.

Step 6: The resulting 3-D CA is the output of this cycle.

Step 7: Perform all the steps k times.

4. RESULTS

Over the years many empirical statistical tests have been developed to determine if there are any hidden correlations in the stream of random numbers. The ultimate goal is to obtain pseudo-random numbers that are like true random numbers. The random number generator has to be tested for its properties. The main two properties that these tests check are uniformity and independence.

The proposed 3-D CA based PRNG is implemented in C++ language. The experiment is repeated 200 times for each rule and the average results obtained are shown in Tables 2, 3, 4. The behavior of the proposed scheme is somewhat similar to a 2-D CA but more secure and random. Also, its complexity

Table 2: Results of the ENT test suite

TEST	3-D CA PRNG	2-D CA PRNG
Chi-Square	127.8524	127.5026
Serial Corr. Coeff.	0.0001	0.0002
Entropy	8.0	7.9999

Table 3: Results of the Diehard test suite

TEST	3-D CA PRNG	2-D CA PRNG
Birthday spacing	0.5997	0.5934
Binary rank 31 × 31	0.5912	0.5709
Binary rank 32 × 32	0.6523	0.6356
Binary rank 6 × 8	0.6554	0.4095
Overlapping permutation	0.5236	0.3699
Bitstream	0.5812	0.5092
OQSO	0.5693	0.4988
OPSO	0.5456	0.5330
Count the ones 01	0.6385	0.5860
Count the ones 02	0.6001	0.5309
DNA	0.5369	0.4971
Minimum Distance	0.5963	0.5093
3DS Spheres	0.5369	0.5120
Overlapping Sum	0.6231	0.5046
Parking Lot	0.5236	0.4389
Squeeze	0.5654	0.5136
Craps	0.6389	0.5092
Runs	0.5966	0.5225

is almost analogous to the 2-D CA based PRNGs. Thus, the novelty in the proposed scheme makes it more suitable for random number generations in both real and non-real time applications. The proposed scheme is tested against the 2-D CA [7] and the results are tested through ENT [23], Diehard [2] and NIST [17] test suits.

The ENT test suite consists of three tests. Each test has its checking mechanism. The results of the ENT test are shown in Table 2.

The Diehard test suite consistes of 18 different tests. Each test has its unique property. A test is considered pass if the value lies between 0.025 and 0.975. The proposed scheme passes all the 18 tests of the Diehard test as shown in Table 3. However, in the case of 2-D CA based PRNG most of the tests are near to the lower limit.

Table 4 shows the results of the NIST test suite. The NIST test suite is one of the most recent test suites developed. It consists of 15 tests out of which this paper has performed 11 of the tests that were relevant to this PRNG.

5. CONTRIBUTIONS

In this research work, a novel 3-D CA based PRNG is proposed. The proposed 3-D CA based PRNG produces random numbers with large linear complexity, better statistical properties, and long period. Also, rules used for updating hte cells are efficient in generating the more random numbers. The random numbers generated using the proposed 3-D CA based generator is a pass to the Diehard, ENT and NIST test suites for randomness. The proposed scheme is

Table 4: Results of the NIST test suite

TEST	3-D CA PRNG	2-D CA PRNG
Frequency	0.5099	0.6035
Block-frequency	0.2134	0.2653
Forward	0.1933	0.2417
Reverse	0.2172	0.2799
Runs	0.3459	0.4271
Long runs	0.5199	0.6034
Rank	0.3561	0.6989
FFT	0.5699	0.7196
Non-periodic templates	0.2199	0.3899
Overlapping templates	0.5999	0.6256
Universal	0.5169	0.6256

compared with the 2-D CA based PRNG and the results obtained prove that the 3-D CA provides better results and is more efficient. The proposed scheme is simple enough to be used with the real and non-real time application. The way the 3-D CA is designed makes it possible for it to be run in parallel and hence achieve much better results. Also, using this PRNG in a harware based PRNG will be a possibility in the near future.

6. REFERENCES

[1] J.-P. Allouche, M. Courbage, and G. Skordev. *Notes on cellular automata*. Citeseer, 2000.

[2] E. D. Berger and B. G. Zorn. Diehard: probabilistic memory safety for unsafe languages. In *Acm sigplan notices*, volume 41, pages 158–168. ACM, 2006.

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

[4] P. P. Chaudhuri. *Additive cellular automata: theory and applications*, volume 1. John Wiley & Sons, 1997.

[5] D. R. Chowdhury, I. Sengupta, and P. P. Chaudhuri. A class of two-dimensional cellular automata and their applications in random pattern testing. *Journal of Electronic Testing*, 5(1):67–82, 1994.

[6] B. Girau and N. Vlassopoulos. Evolution of 2-dimensional cellular automata as pseudo-random number generators. *Cellular Automata*, pages 611–622, 2012.

[7] S.-U. Guan and S. Zhang. An evolutionary approach to the design of controllable cellular automata structure for random number generation. *IEEE Transactions on Evolutionary Computation*, 7(1):23–36, 2003.

[8] P. D. Hortensius, R. D. McLeod, and H. C. Card. Parallel random number generation for vlsi systems using cellular automata. *IEEE Transactions on Computers*, 38(10):1466–1473, 1989.

[9] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(8):842–859, 1989.

[10] D. E. Knuth. The art of programming, vol. 2, semi-numerical algorithms, 1981.

[11] P. L'Ecuyer. Uniform random number generators: a review. In *Proceedings of the 29th conference on Winter simulation*, pages 127–134. IEEE Computer Society, 1997.

[12] G. Marsaglia. The structure of linear congruential sequences. *Applications of number theory to numerical analysis*, pages 249–285, 1972.

[13] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[14] F. Panneton, P. L'ecuyer, and M. Matsumoto. Improved long-period generators based on linear recurrences modulo 2. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):1–16, 2006.

[15] J. B. Plumstead. Inferring a sequence produced by a linear congruence. In *CRYPTO*, pages 317–319, 1982.

[16] W. H. Press, B. Flannery, S. Teukolsky, and W. Vetterling. Numerical recipes in c. cambridge university press 1992. Technical report, ISBN 0-521-43108-5.

[17] B. Y. Ryabko, V. Stognienko, and Y. I. Shokin. A new test for randomness and its application to some cryptographic problems. *Journal of statistical planning and inference*, 123(2):365–376, 2004.

[18] J. Schiff. Introduction to cellular automata, 2005.

[19] S.-H. Shin, D.-S. Kim, and K.-Y. Yoo. A 2-dimensional cellular automata pseudorandom number generator with non-linear neighborhood relationship. *Networked Digital Technologies*, pages 355–368, 2012.

[20] M. Tomassini, M. Sipper, and M. Perrenoud. On the generation of high-quality random numbers by two-dimensional cellular automata. *IEEE Transactions on computers*, 49(10):1146–1151, 2000.

[21] P. Tsalides, T. York, and A. Thanailakis. Pseudorandom number generators for vlsi systems based on linear cellular automata. *IEE Proceedings E (Computers and Digital Techniques)*, 138(4):241–249, 1991.

[22] A. Vaskova, C. López-Ongil, E. San Millán, A. Jiménez-Horas, and L. Entrena. Accelerating secure circuit design with hardware implementation of diehard battery of tests of randomness. In *On-Line Testing Symposium (IOLTS), 2011 IEEE 17th International*, pages 179–181. IEEE, 2011.

[23] J. Walker and A. Ent. Pseudorandom number sequence test program, 1998. *See http://www. fourmilab. ch/random*, 2013.

[24] S. Wolfram. Random sequence generation by cellular automata. *Advances in applied mathematics*, 7(2):123–169, 1986.

[25] S. Wolfram et al. Cellular automata as models of complexity. *Nature*, 311(5985):419–424, 1984.