Pseudo-random sequence generators based on cellular automata and bent functions[1]

FRANCISCO J. GARCÍA[(a)] VERÓNICA REQUENA[(b)2], VIRTUDES TOMÁS[(b)3]
[(a)]Departament de Fonaments de l'Anàlisis Econòmica
[(b)]Departament de Ciència de la Computació i Intel·ligència Artificial
Universitat d'Alacant
Campus de Sant Vicent del Raspeig
Ap. correus 99, E-03080 Alacant
SPAIN
francisco.garcia@ua.es, vrequena@dccia.ua.es, vtomas@dccia.ua.es

Abstract:
In this article we construct different pseudo-random sequences using cellular automata where the local transition functions are based on balanced functions which are obtained from bent functions.

Key-words: Pseudo-random sequence, cellular automata, bent function, balanced function.

1 Introduction

Pseudo-random number sequences are needed in many important applications, such as Monte Carlo techniques, Brownian dynamics, and stochastic optimization methods. Cellular automata (CA) offer a number of advantages over other methods as random number generators such as algorithmic simplicity and easy hardware implementation.

Cellular automata were first introduced by von Neumann and later by Wolfram [11] as simple models for physical, biological and computational systems. CA have previously been used as encrypting devices by Wolfram [13] and by Nandi, Kar and Chaudhuri [3]. Gutowitz [2] and Guam [1] used CA for public-key cryptography.

The pseudo-random sequence generator based on CA has been extensively studied in the last decades. In 1986, Wolfram [13] first applied CA in pseudorandom number generation. After, other authors as Hortensius, Tsalides, Sipper and Perrenoud used the CA to generate the pseudorandom sequences used in cryptography [6, 9, 14]. But those methods are based on artificial methods to construct CA rules. The main disadvantage of those methods is that they are involved in large tedious work. Another valid method is introduced by Sipper and Tomassini [6], they used genetic algorithm to find the best CA randomizer rules automatically. A series of research work has been done to generate pseudo-random sequences.

In this article, we construct pseudo-random sequence generator using a one-dimensional finite CA where the local transition functions are based on balanced functions which are obtained from bent functions. The use of bent functions can generate more security due to their good cryptographic properties.

The rest of the paper is organized as follows. In Section 2 we introduce some basic concepts about cellular automata, pseudo-random sequences generators, and bent functions and the notation we will use. In Section 3 we describe three different pseudo-random sequence generators based on cellular automata and in Section 4 we present some of our main results. Finally, we provide some conclusions in Section 5.

2 Preliminaries

In this section we introduce the main concepts related to cellular automata, pseudo-random sequence generators and bent functions.

2.1 Cellular Automata

Cellular automata were originally conceived by Ulam and von Neumann in the 1940s to provide a formal framework for investigating the behavior of complex, extended systems [10, 7]. A cellular automaton is a discrete dynamical system. Space, time, and the states of the system are discrete. Each point in a regular spatial lattice, called a cell, can have any one of a finite number of states. The states of the cells in the lattice are updated according to a local rule.

[1]This work was partially supported by Spanish grant MTM2005-05759.
[2]The work of this author was supported by a grant of the Vicerectorat d'Investigació, Desenvolupament i Innovació of the Universitat d'Alacant for PhD students.
[3]The work of this author was supported by a grant of the Vicerectorat d'Investigació, Desenvolupament i Innovació of the Universitat d'Alacant for PhD students.

That is, the state of a cell at a given time depends only on its own state one time step previously, and the states of its nearby neighbors at the previous time step (see [12, 8]). All cells on the lattice are updated synchronously. Thus the state of the entire lattice advances in discrete time steps.

The most used CA are finite and one-dimensional. More precisely, a **one-dimensional finite** CA can be defined as a 4-tuple $\mathcal{A} = (C, S, V, f)$. C is the cellular space formed by a linear array of m cells; each cell is denoted by $\langle i \rangle$, $0 \leq i \leq m-1$. S is the (finite) state set, that is, the set of all possible values of the cells; usually, if the CA considered is finite with k states, then $S = \mathbb{Z}_k$. V is the set of cells which states in the instant t influence in state of the cell considered in the instant $t+1$; in this work and for the particular case of the one-dimension CA, we will consider for every cell $\langle i \rangle \in C$, its neighborhood V_r as the ordered set given by

$$V_r = \{\langle i-r \rangle, \ldots, \langle i \rangle, \ldots, \langle i+r \rangle\}.$$

Moreover, the **local transition function** $f : S^{(2r+1)} \rightarrow S$ is the function determining the evolution of the CA throughout the time, i.e., the changes of the states of every cell taking the states of its neighbors into account; hence, if $a_i^{(t)} \in S$ stands for the state of the cell $\langle i \rangle$ at time t, and $V_i^{(t)}$ is the set of the states of the cell $\langle i \rangle$ at that time; the next state of the cell is given by

$$a_i^{(t+1)} = f\left(a_{i-r}^{(t)}, \ldots, a_i^{(t)}, \ldots, a_{i+r}^{(t)}\right) \qquad (1)$$

As the cellular space is finite, boundary conditions must be established in order to ensure that the evolution of the cellular automata is well-defined.

The set of states of all cells at time t is called the **configuration at time** t and it is represented by the vector

$$C^{(t)} = \left(a_0^{(t)}, a_1^{(t)}, \ldots, a_{m-1}^{(t)}\right) \in S^n.$$

In particular, $C^{(0)}$ is the **initial configuration**. Hence, the **evolution** of $\mathcal{A}$ is the following sequence

$$\left(C^{(1)}, C^{(2)}, \ldots\right).$$

If we denote by $\mathcal{C}$ the set of all possible configurations of $\mathcal{A}$, then the global function of $\mathcal{A}$ is a linear transformation, $\Phi : \mathcal{C} \rightarrow \mathcal{C}$, that yields the configuration at the next time step during the evolution of the CA; that is, $C^{(t+1)} = \Phi(C^{(t)})$. If Φ is bijective then there exists another cellular automaton, $\mathcal{A}^{-1}$, called its **inverse**, whose global function is Φ^{-1}. When such inverse cellular automaton exists, $\mathcal{A}$ is called **reversible** and the evolution backwards is possible (see [][24]). In general, the evolution of a CA considers that the state of every cell at time $t+1$ depends on the state of its neighborhood at time t, $V_r^{(t)}$. Nevertheless, one can consider that this evolution also depends on the states of other cells at times $t+1, t+2$, etc. In this case, the transition function given in (1) can be represented in the following way

$$a_i^{(t+1)} = \sum_{h=0}^{k} f^{(t-h)} V_r^{(t-h)}$$

where each $f^{(t-h)}$ is a specific local transition function.

2.2 Pseudo-random sequence generator

Definition 1: A **random bit generator** is a device or algorithm, which outputs a sequence of statistically independent and unbiased binary digits.

A random bit generator can be used to generate uniformly distributed random numbers. Random sequence generation is difficult to do in computer, since computers are deterministic devices. Thus, is the same random generator is run twice on computer, identical results are received. Random sequence generator are in used, but they can be difficult to generate. Because of this difficulties, random sequence generator in a computer is usually only pseudo-random sequence generator (see [15]).

Definition 2: A **pseudo-random bit generator** (PRBG) is a deterministic algorithm which, given a truly random binary sequence of length m, outputs a binary sequence of length $k \gg m$ which "appears" to be random. The input to the PRBG is called the **seed** and the output is called a **pseudo-random bit sequence** or **pseudo-random sequence**.

The output of a PRBG is not random, in fact the number of possible output sequences of length k is at most all small fraction $2m/2k$, as the PRBG produces always the same output sequence for one (fixed) seed. The motivation for using a PRBG is that it might be too expensive to produce true random numbers of length k, e.g. by coin flipping, so just a smaller amount of random bits is produced and then a pseudo-random bit sequence is produced out of the m truly random bits. Good random properties of the generator are convenient to prevent statistical attacks; but

moreover, it is necessary that the generator must be cryptographically secure.

In order to gain confidence in the "randomness" of a pseudo-random sequence, statistical tests are conducted on the produced sequences. In our study, we have considered the seven main statistic tests which allow us to check, if a generated random or pseudo-random sequence inhibits certain statistical properties

1. **Frequency Test (Monobit Test)**: The focus of the test is the proportion of zeros and ones for the entire sequence. The purpose of this test is to determine whether the number of ones and zeros in a sequence are approximately the same as would be expected for a truly random sequence. The test assesses the closeness of the fraction of ones to ?, that is, the number of ones and zeros in a sequence should be about the same. All subsequent tests depend on the passing of this test; there is no evidence to indicate that the tested sequence is non-random.

2. **Serial Test (Two-bit Test)**: The purpose of the serial test is to determine if the number of occurrences of 00, 01, 10 and 11 as subsequences of the tested string are approximately the same, as expected from a random bit string.

3. **Poker Test**: Let p be a positive integer such that $\left\lfloor \frac{k}{q} \right\rfloor \geq 5 \cdot (2^q)$, and let $r = \left\lfloor \frac{k}{q} \right\rfloor$. Divide the sequence s into r non-overlapping parts each of length p, and let n_j be the number of occurrences of the jth type of sequence of length p, $1 \leq j \leq 2^p$. The poker test determines whether the sequences of length p each appear approximately the same number of times in s, as would be expected for a random sequence.

4. **Runs Test**: The runs distribution test compare the distribution of the numbers of ones (blocks) and zeros (gaps) with that expected under randomness. The runs test can be used to support results from the previous tests. Failure of the runs test indicates that there is a bad distribution of runs lengths or that there are no runs recorded above a certain length that are expected top occur for streams of the sample size. The zero frequencies recorded will result in a higher chi-square statistic thus giving a smaller significance probability.

5. **Autocorrelation Test**: The purpose of this test is to check for correlations between the sequence s and (non-cyclic) shifted versions of it.

6. **The Linear Complexity Text** The linear complexity test checks of the minimum of bits needed to reconstruct the whole stream. Every finite stream, s, can be produced by a LFSR. The length of the shortest LFSR which will produce the stream is said to be the linear complexity of the stream. If the value of LC is L then $2L$ consecutive terms can be used to reconstruct the whole sequence using the Berlekamp Massey algorithm. Hence, in order to avoid stream reconstruction, the LC value should be large.

7. **Maurer's Universal Test**: The basic idea behind Maurer's universal test is that it should not be to possible to significantly compress the output sequence of a random bit generator. Thus, if a sample output sequence s of a bit generator can be significantly compressed, the generator should be rejected as being defective. Instead of actually compressing the sequence s, the universal statical test computes a quantity that is related to the length of the compressed sequence.

The above descriptions just give the basic ideas of the tests.

2.3 Bent functions

A Boolean function of n variables is a mapping $g : \mathbb{Z}_2^n \longrightarrow \mathbb{Z}_2$. We call the **truth table** of g the $(0,1)$-sequence of length 2^n given by

$$\boldsymbol{\xi}_g = (g(\boldsymbol{u}_0),\ g(\boldsymbol{u}_1),\ \ldots,\ g(\boldsymbol{u}_{2^n-1})).$$

The **Hamming distance** between two $(0,1)$-sequences $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, denoted by $d(\boldsymbol{\alpha}, \boldsymbol{\beta})$, is the number of postions where the two sequences differ. If $g(\boldsymbol{x})$ and $h(\boldsymbol{x})$ are Boolean functions and $\boldsymbol{\xi}_g$ and $\boldsymbol{\xi}_h$ are the corresponding truth table, the **Hamming distance** between g and h, denoted by $d(g,h)$, is the Hamming distance between the $(0,1)$-sequences $\boldsymbol{\xi}_g$ and $\boldsymbol{\xi}_h$.

A $(0,1)$-sequence is **balanced** if it contains an equal number of 0s and 1s; so, a Boolean function is **balanced** if its truth table is balanced.

We say that $g \in \mathcal{B}_n$ is an **affine function** if it takes the form

$$g(\boldsymbol{x}) = \langle \boldsymbol{a}, \boldsymbol{x} \rangle \oplus b$$

where $\boldsymbol{a} \in \mathbb{Z}_2^n$ and $b \in \mathbb{Z}_2$. If $b = 0$, we say that f is a **linear function**.

We define the **nonlinearity** of a function $g \in \mathcal{B}_n$ as

$$\mathrm{NL}(g) = \min\{d(g,\varphi) \mid \varphi \in \mathcal{A}_n\}$$

where $\mathcal{A}_n$ is the set of all affine functions. The nonlinearity of g is upper bounded (see [5]) by

$$\mathrm{NL}(f) \leq 2^{n-1} - 2^{\frac{n}{2}-1}.$$

We call **bent functions** the Boolean functions that achieve the maximum nonlinearity (see [5]). Consequently, bent functions only exist for n even.

The following result (see [4, 5]), that we quote for further references, gives us a characterization of a bent function.

Theorem 1: *Let $g(\boldsymbol{x})$ be a Boolean function of n variables. The following statements are equivalent:*

1. *$g(\boldsymbol{x})$ is a bent function.*
2. *For any $\boldsymbol{a} \in \mathbb{Z}_2^n \setminus \{\mathbf{0}\}$, the Boolean function $g(\boldsymbol{x}) \oplus g(\boldsymbol{a} \oplus \boldsymbol{x})$ is balanced.*
3. *$1 \oplus g(\boldsymbol{x})$ is also bent function; moreover, this is the complementary function of g.*

3 Models

In this section we present three different pseudo-random sequence generators based on cellular automata where the local transition function, f, is generate from bent functions.

3.1 Model 1

In the first model of pseudo-random sequence generator, we have considered the CA defined by the 4-tuple $\mathcal{A} = (C, S, V, f)$, where

1. C is the cellular space formed by a linear array of $m = 256$ cells. Initially, we fix a random initial configuration vector

$$C^{(0)} = \left(a_0^{(0)}, a_1^{(0)}, \ldots, a_{m-1}^{(0)}\right).$$

2. S is the finite set of states k, where $S = \mathbb{Z}_k$, with $k = 10000$.
3. For every cell $\langle i \rangle \in C$, we have taken V_2, i.e, the set given by

$$V_2 = \{\langle i-2 \rangle, \langle i-1 \rangle, \langle i \rangle, \langle i+1 \rangle, \langle i+2 \rangle\}$$

.

4. The **local transition function**, f, is based on bent functions in the following way. By Theorem 1 we know that the function defined by $g(\boldsymbol{x}) \oplus g(\boldsymbol{a} \oplus \boldsymbol{x})$, for any $\boldsymbol{a} \in \mathbb{Z}_2^n \setminus \{\mathbf{0}\}$ being g a bent function, is balanced. So, in this case, we take as f this balanced function defined by bent functions of 4 variables and in some specific cases we take the complementary function, $1 \oplus g(\boldsymbol{x})$.

   It is well know that in the case $n = 4$, the number of bent functions is 896, although we only consider the half, 448, so that, these are the main functions, being the rest the complementary functions.

   In each state t for $t > 0$, we need to take a bent function, $g(\boldsymbol{x})$, and a value $\boldsymbol{a}$ to construct the function $g(\boldsymbol{a} \oplus \boldsymbol{x})$. We fix the bent function $g(\boldsymbol{x})$ in the following way. Take the decimal representation of the configuration at time $t-1$, compute it module 448 and add 1; the resulting number will be in the range of the bent functions, and we consider the bent function associated to this number. We distinguish two cases to fix the function f of our CA.

   Let $\boldsymbol{a} = t \mod 2^n$.

   - If $\boldsymbol{a} \neq 0$, then $f(\boldsymbol{x}) = g(\boldsymbol{x}) \oplus g(\boldsymbol{a} \oplus \boldsymbol{x})$.
   - If $\boldsymbol{a} = 0$, then $f(\boldsymbol{x}) = 1 \oplus g(\boldsymbol{x})$.

Then we keep the values $C^{(k)}(n/2)$ which generate the pseudo-random sequence.

3.2 Model 2

In this model and the next, we have taken the same C, S, V, but change the way to choose the local transition function f.

Initially we fix a bent function of 4 variables, $g(\boldsymbol{x})$, and a value of $\boldsymbol{a}$ inside the range $[1, 2^n - 1]$. Here, we do not distinguish different cases according to the state to choose f; however we always take as local transition function the balanced function defined by the bent function fixed; that is, $f(\boldsymbol{x}) = g(\boldsymbol{x}) \oplus g(\boldsymbol{a} \oplus \boldsymbol{x})$.

With these parameters we generate a different pseudo-random sequence for each bent function and for each value of $\boldsymbol{a}$. As in the previous model, we keep the values $C^{(k)}(n/2)$ which generate our pseudo-random sequence.

3.3 Model 3

Finally, this model perform a mixed model using the previous ones.

First, we fix a bent function of 4 variables, $g(\boldsymbol{x})$ and for each function we generate a different pseudo-random sequence.

We have a different local transition function f according to the value $\boldsymbol{a}$ chosen. We take $\boldsymbol{a}$ in the fol-

lowing way. For $l = 0, 1, \ldots, \lfloor \frac{k}{2^n} \rfloor$

$$a = t \mod l(2^n - 1)$$

At the end, we obtain of the same way the pseudo-random sequence generator.

4 Results

In this section we report the main results that we have obtained from the three models of pseudo-random sequence generators explained in the previous section. We have considered random seed vectors for the different models. As a minimum security requirement the length k of the seed to a PRBG should be large enough to make brute-force search over all seeds infeasible for an attacker. The length of the seeds that we have considered in our work is $m = 256$.

In the first model, we use all bent functions and all values of $\boldsymbol{a}$ for each seed. In this case, all the sequences that we have obtained for the different seeds pass all our tests.

In the second model, for each bent function we fix a value for $\boldsymbol{a}$ and we check some random seeds for these. In this model and the next, we have taken as valid functions those that at least six of the ten seeds pass all the tests. Let us see some particular cases. In the following table we can observe the percentage of valid bent functions for ours, for some values of the $\boldsymbol{a}$s.

$\boldsymbol{a}$	percentage of valid bent functions
1	58%
2	66%
3	56%
9	61%
15	64%

The seeds that we took in these examples were random, therefore we have different seeds in each case. In a future work we will study the behaviour of the bent functions for the different values of $\boldsymbol{a}$ using the same seeds.

Studying all cases for the different values of $\boldsymbol{a}$, we can bring that around 63% of the bent function pass all the tests.

In the last model, for each bent function we check some seeds. In this case we do not distinguish between the values of $\boldsymbol{a}$ because for each bent function and for each seed we take all the values of $\boldsymbol{a}$. The results that we have obtained are that around 35% of the bent function pass all the tests.

5 Conclusions

If we observe the previous results it seems to be that the best model to generate pseudo-random sequences based on cellular automata using bent functions is the first model, where we use all bent functions and the different values for $\boldsymbol{a}$ for each seed taken. However, the others models are worse. Consequently, we can conclude that use a unique bent function to generate the local transition function of a CA is worse than work with all or with a big number of bent functions in the different states.

Now, our research is based on the study of some particular bent functions that have had good behaviour in all the models that we have explained, and then to observe which generate the best results.

References:

[1] P. GUAM. Cellular automaton public key cryptosystems. *Complex Syst.*, 1: 51–56 (1987).

[2] H. GUTOWITZ. Cryptography with dynamical systems. In E. GOLES, N. BOCCARA(eds.), *Cellular Automata and Cooperative Phenomena*, Kluwer Academic Publishers, Dordrecht, 1993.

[3] S. NANDI, B. K. KAR and P. P. CHAUDHURI. Theory and applications of cellular automata in cryptography. *IEEE Transaction on Compututers*, 43(12): 1346–1357 (1994).

[4] J. SEBERRY and X.-M. ZHANG. Constructions of bent functions from two known bent functions. *Australasian Journal of Combinatorics*, **9**: 21–35 (1994).

[5] J. SEBERRY, X.-M. ZHANG and Y. ZHENG. Nonlinearity and propagation characteristics of balanced Boolean functions. *Information and Computation*, 119: 1–13 (1995).

[6] M. SIPPER and M. TOMASSINI. Computation in artificially evolved, non–uniform cellular automaton. *Theoretical Computer Science*, 217(1): 81–98 (1999).

[7] M. SIPPER. Machine Nature: The Coming Age of Bio-Inspired Computing. *McGraw-Hill*, New York, 2002

[8] T. TOFFALI and N. MARGOLUS. Cellular Automata Machines. *The MIT Press*, Cambridge, Massachusetts, 1987.

[9] M. TOMASSINI and M. PERRENOUD. Crytography with cellular automata. *Applied Soft Computing Journal*, 1(2): 151–160 (2001).

[10] J. VON NEUMANN. Theory of Self-Reproducing Automata. *University of Illinois Press*, Illinois, 1966. Edited and Completed by A. W. Burks.

[11] S. WOLFRAM. Statistical mechanics of cellular automata. *Rev. Mod. Phys.*, 55(3): 601–644 (1983).

[12] S. WOLFRAM. Universality and Complexity in Cellular Automata. *Physica D*, 10:1-35 (1984).

[13] S. WOLFRAM. Crytography with cellular automata. In *Advances in Cryptology — Crypto'85*, Vol. 218 of *Lecture Notes in Computer Science*, pages 429–432. Springer, Heidelberg, 1986.

[14] S. WOLFRAM. A New Kind of Science. *Wolfram Media, Inc.*, Illinois, 2002.

[15] K. ZENG, C. H. YANG, D. Y. WEI and T. R. N. RAO. Pseudorandom Bit Generators in Stream-Cipher Cryptography. *IEEE Computer Magazine*, 24(2): 8–17 (1991).