Evolutionary Design of Pseudorandom Sequence Generators based on Cellular Automata and Its Applicability in Current Cryptosystems

David Delgado
Comp. Engineering,
National University of Colombia
and Economic Inform. Centre,
Banco de la República
ddelgach@banrep.gov.co

David Vidal
Comp. Engineering,
National University of Colombia
dfvidalo@unal.edu.co

German Hernandez
Comp. Engineering,
National University of Colombia
gjhernandezp@unal.edu.co

ABSTRACT

In this work, a genetic algorithm is used to find cellular automata rules that make cellular automata behave like good pseudorandom sequence generators. Pseudorandom sequence generators based on one-dimensional cellular automata with non-homogeneous rules and arbitrary neighbors are proposed. The fitness function combines entropy measures and standard statistical tests for random sequences. The generators found are statistically compared to some well-known pseudorandom sequences generators.

Track Category

Real-World Applications (RWA).

Categories and Subject Descriptors

E.3 **[Data Encryption]:** Standards (e.g. DES, PGP, RSA).

F.1.1 **[Models of computation]:** Unbounded-action devices (cellular automata, circuits, network of machines).

G.3 **[Probability and Statistics]:** Random Number Generation.

General Terms

Genetic Algorithms, Design, Experimentation, Security.

Keywords

Cryptography, Pseudorandom Generator, Cellular automata, Genetic algorithm, Statistical test.

1. INTRODUCTION

Cellular automata have become an attractive alternative for pseudorandom number generation. Due to its simplicity, good generators can be easily implemented in hardware devices such as FPGA[3]. Pseudorandom generators play and important rule in several current cryptosystems. However, generating pseudorandom sequences is a hard problem because such sequences must seem random and must be generated by a deterministic algorithm.

The main objective of this study is to analyze the effect of varying the neighborhood shape in the performance of CA like pseudorandom generators. For this task, in this paper is studied CA with a number between one and five arbitrary neighbors.

2. RANDOM SEQUENCE GENERATION AS AN OPTIMIZATION PROBLEM

In cryptography, for the stream cipher case, where a plaintext (written like a bit stream) is transformed into a ciphertext through bit to bit sum with a pseudorandom sequence of bits, the security of this method depends on how similar is the generated sequence with respect to a truly random sequence.

A cellular automaton (CA) is a computational device composed of a uniform cell array and finite rules set that are applied to each cell. Many cellular automata exhibit a global chaotic and unpredictable behavior and, for that reason, they have been proposed to be used as pseudorandom sequence generators.

To solve the problem of choosing the rules that generate the most unpredictable global behavior, Tomassini[4] proposed an evolutionary technique called *Cellular Programming* (CP).

Recent studies have considered cellular automata with properties like:

- A large number of neighbors that tends to increase the size of the asymptotic limiting cycle but this carries the problem that the rule space size becomes so large that is impossible to explore it exhaustively. Therefore, evolutionary techniques become suitable to deal with this problem.

- Asynchronic updating that tends to avoid reaching asymptotic limiting cycles.

- Some neighborhood shapes as is proposed in Shackleford[3] and Koza[1] . This reduces local dependency that avoids long-term pattern formation.

In this work, it has been considered one-dimensional cellular automata with non-homogeneous local rules, in which each cell has between one and five arbitrary neighbors.

3. THE EVOLUTIONARY ALGORITHM

The genetic algorithm to find good rules is a modified version of the CP[4]. The main difference with this technique is:

- A fitness function that includes a term that is computed based on some statistical tests, in addition to the entropy term used in cellular programming. The objective is to detect initially the rules that don't pass the tests and discard them easily.

$$\textbf{Fitness} = H_s(A) + \sum_{i=1}^{k} w_i * d_i(A)$$

where:

A is the temporal configuration of the cell

$H_s(A)$ is the entropy of blocks with size *s* in *A*

w_i is the weight that is assigned to each statistical test i

$d_i(A)$ = 1 if the sequence passes the test i

= 0 if the sequence fails the test i

A general description of the genetic algorithm is as follows:

First, the fitness function is calculated for each cell. Next, the fitness of the cell is compared to the fitness of the neighbors, and according to the number of them with better fitness, the following actions are taken: If this number is zero, the rule does not change. If the number is 1, the rule and neighborhood shape of the better neighbor is mutated and copied. If this number is 2 or more, two neighbors with better fitness are chosen at random and a crossover between such neighbors is performed.

Four crossover operations were included. Considering two rules represented by the binary string corresponding to the rules numbers, and their length representations are 2^n and 2^m, we have:

- Crossover 1: *If* $n<m$, the crossover point is chosen in the shorter string and if $n=m$, the common single-point crossover is used.
- Crossover 2: *If* $n<m$, the two crossover point are chosen in the shorter string and if $n=m$, the common double-point crossover is used.
- Crossover 3: First, crossover 2 is performed, then the two binary strings are concatenated, and, finally, the resulting string is split in two pieces at random, but ensuring that the lengths of the two pieces are powers of 2 (feasible rules).
- Crossover4: Let the rules length representation be 2^n and 2^m, with $n < m$, then we pick at random one of the consecutive subsegments of size 2^n in the string of size 2^m and swap the bits of that subsegment with the shorter string.

4. EXPERIMENTS

The first observation was that the rules converged almost the time and for most of the cells to 5-neighbor rules.

With the genetic algorithm described in previous section, 20 rules were the most frequently found in the best pseudorandom generators based on cellular automata and then, each one of the twenty rules was used to build a synchronous homogeneous cellular automaton with 128 cells and this was used to generate 80 samples of 10000 bits. These samples were obtained starting from a random initial configuration and then concatenating different output configurations. These samples were evaluated with the NIST[2] statistical test suite using the recommended parameters given in the suite specification.

The rule that had the best performance was the rule 2572018122. This rule was used to generate 10 samples with 1000000 bits each one and they were compared with some well-known pseudorandom generators used in current cryptosystems (like the ANSI X9.17, the Blum Blum Shub generator, and the G-DES) and the results obtained show that sequences generated are similar with respect to the ANSI X9.17 and slightly better than the Blum Blum Shub and the linear congruential generator. However, to confirm this, it is necessary to make a study with bigger samples. Several space –time diagrams were generated using the DDLab tool[5] and they show how difficult is to detect patterns with the rule 2572018122 applied to an automaton with 5 arbitrary neighbors.

5. CONCLUSIONS

- It has been proposed a new method for generating pseudorandom sequences based on one-dimensional cellular automata with 5 neighbors that can be or not adjacent to each cell. For selecting the optimal rules it was used a genetic algorithm with a new fitness function and crossovers.
- Sequences generated by the rule 2572018122 are undistinguishable (in limited computing time) from truly random sequences, from a statistical point of view because they almost pass all the tests applied to them. Besides, this method is appealing because it is easy to generate sequences using the simple structure of transition rules that characterizes the CA.

6. ACKNOWLEDGEMENTS

To the engineer German Hernandez for his guidance, his useful comments, his worthy collaboration, his interest for our research and his trust in ours, and to the engineer Fernando Niño for his support along our research.

7. REFERENCES

[1] KOZA, John R., *Genetic Programming*, MIT press, Massachusetts, 1994.

[2] RUKHIN, Andrew, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications*, NIST Special Publication 800-22, 2001.

[3] SHACKLEFORD, Barry. *FPGA Implementation of Neighborhood-of-Four Cellular Automata Random Number Generators*, Hewlett-Packard Laboratories, 2002.

[4] TOMASSINI, Marco. *Generating high-quality random numbers in parallel by cellular automata*, Future Generation Computer Systems, vol. 16, pp. 291–305, 1999.

[5] WUENSCHE A., y LESSER M., *The Global Dynamics of Cellular Automata*, Santa Fe Institute Studies in the Sciences of Complexity, Addison-Wesley, MA, 1992.