Traversing a n-cube without Balanced Hamiltonian Cycle to Generate Pseudorandom Numbers

J.-F. Couchot, P.-C. Heam, C. Guyeux, Q. Wang, and J. M. Bahi
FEMTO-ST Institute, University of Franche-Comté, France
College of Automation, Guangdong University of Technology, China
{jean-francois.couchot,pierre-cyrille.heam,christophe.guyeux,
jacques.bahi}@univ-fcomte.fr,
wangqianxue@gdut.edu.cn

December 18, 2014

Abstract

This article presents a new class of Pseudorandom Number Generators. The generators are based on traversing a n-cube where a Balanced Hamiltonian Cycle has been removed. The construction of such generators is automatic for small number of bits, but remains an open problem when this number becomes large. A running example is used throughout the paper. Finally, first statistical experiments of these generators are presented, they show how efficient and promising the proposed approach seems.

1 Introduction

Many fields of research or applications like numerical simulations, stochastic optimization, or information security are highly dependent on the use of fast and unbiased random number generators. Depending on the targeted application, reproducibility must be either required, leading to deterministic algorithms that produce numbers as close as possible to random sequences, or refused, which implies to use an external physical noise. The former are called pseudorandom number generators (PRNGs) while the latter are designed by truly random number generators (TRNGs). TRNGs are used for instance in cypher keys generation, or in hardware based simulations or security devices. Such TRNGs are often based on a chaotic physical signal, may be quantized depending on the application. This quantization however raises the problem of the degradation of chaotic properties.

The use of PRNGs, for its part, is a necessity in a large variety of numerical simulations, in which responses of devices under study must be compared using the same "random" stream. This reproducibility is required too for symmetric encryption like one-time pad, as sender and receiver must share the same pad. However, in that situation, security of the pseudorandom stream must be mathematically proven: an attacker must not be able to computationally distinguish a pseudorandom sequence generated by the considered PRNG with a really random one. Such cryptographically secure pseudorandom number generators are however only useful in cryptographic contexts, due to their slowness resulting from their security.

Other kind of properties are desired for PRNGs used in numerical simulations or in programs that embed a Monte-Carlo algorithm. In these situations, required properties are speed and random-like profiles of the generated sequences. The fact that a given PRNG is unbiased and behaves as a white noise is thus verified using batteries of statistical tests on a large amount of pseudorandom numbers. Reputed and up-to-date batteries are currently the NIST suite [2], and DieHARD [5]. Finally, chaotic properties can be

desired when simulating a chaotic physical phenomenon or in hardware security, in which cryptographical proofs are not realizable. In both truly and pseudorandom number generation, there is thus a need to mathematically guarantee the presence of chaos, and to show that a post-treatment on a given secure and/or unbiased generator can be realized, which adds chaos without deflating these desired properties.

This work takes place in this domain with the desire of automatically generating a large class of PRNGs with chaos and statistical properties. In a sense, it is close to [1] where the authors shown that some Boolean maps may be embedded into an algorithm to provide a PRNG that has both the theoretical Devaney's chaos property and the practical property of succeeding NIST statistical battery of tests. To achieve this, it has been proven in this article that it is sufficient for the iteration graph to be strongly connected, and it is necessary and sufficient for its Markov probability matrix to be doubly stochastic. However, they do not purpose conditions to provide such Boolean maps. Admittedly, sufficient conditions to retrieve Boolean maps whose graphs are strongly connected are given, but it remains to further filter those whose Markov matrix is doubly stochastic. This approach suffers from delaying the second requirement to a final step whereas this is a necessary condition. In this position article, we provide a completely new approach to generate Boolean functions, whose Markov matrix is doubly stochastic and whose graph of iterations is strongly connected. Furthermore the rate of convergence is always taken into consideration to provide PRNG with good statistical properties.

This research work is organized as follows. It firstly recall some preliminaries that make the document self-contained (Section 2), The next section (Section 3) shows how the problem of finding some kind of matrices is moved into the graph theory. Section 4 is the strongest contribution of this work. It presents the main algorithm to generate Boolean maps with all the required properties and proves that such a construction is correct. Statistical evaluations are then summarized in Section 5. Conclusive remarks, open problems, and perspectives are finally provided.

2 Preliminaries

In what follows, we consider the Boolean algebra on the set $\mathbb{B} = \{0,1\}$ with the classical operators of conjunction '.', of disjunction '+', of negation '-', and of disjunctive union $\oplus$. Let n be a positive integer. A *Boolean map* f is a function from the Boolean domain to itself such that $x = (x_1,\ldots,x_n)$ maps to $f(x) = (f_1(x),\ldots,f_n(x))$. Functions are iterated as follows. At the t^{th} iteration, only the s_t-th component is "iterated", where $s = (s_t)_{t\in\mathbb{N}}$ is a sequence of indices taken in $[\![1;n]\!]$ called "strategy". Formally, let $F_f : [\![1;n]\!] \times \mathbb{B}^n$ to $\mathbb{B}^n$ be defined by

$$F_f(i,x) = (x_1,\ldots,x_{i-1},f_i(x),x_{i+1},\ldots,x_n).$$

Then, let $x^0 \in \mathbb{B}^n$ be an initial configuration and $s \in [\![1;n]\!]^{\mathbb{N}}$ be a strategy, the dynamics are described by the recurrence

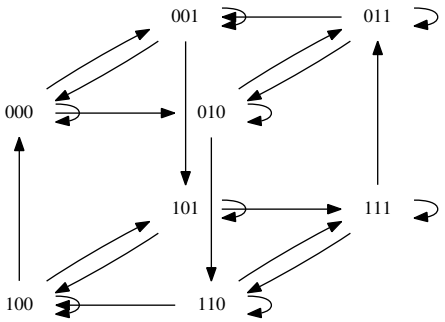
$$x^{t+1} = F_f(s_t,x^t). \tag{1}$$

Let be given a Boolean map f. Its associated *iteration graph* $\Gamma(f)$ is the directed graph such that the set of vertices is $\mathbb{B}^n$, and for all $x \in \mathbb{B}^n$ and $i \in [\![1;n]\!]$, the graph $\Gamma(f)$ contains an arc from x to $F_f(i,x)$.

It is easy to associate a Markov Matrix M to such a graph $G(f)$ as follows: $M_{ij} = \frac{1}{n}$ if there is an edge from i to j in $\Gamma(f)$ and $i \neq j$; $M_{ii} = 1 - \sum_{j=1,j\neq i}^{n} M_{ij}$; and $M_{ij} = 0$ otherwise.

Running example. *Let us consider for instance $n = 3$. Let $f^* : \mathbb{B}^3 \to \mathbb{B}^3$ be defined by $f^*(x_1,x_2,x_3) = (x_2 \oplus x_3, x_1 \oplus \overline{x_3}, \overline{x_3})$. The iteration graph $\Gamma(f^*)$ of this function is given in Figure 1(a) and its Markov matrix is given in Figure 1(b).*

The *mixing time* [4] is one of the usual metrics that gives how far the rows of a Markov matrix converge to a specific distribution. It defines the smallest iteration number that is sufficient to obtain a deviation lesser than a given ε for each rows of such kind of matrices.



(a) Iteration Graph $\Gamma(f^*)$ of the function f^*

$$\frac{1}{3}\begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

(b) Markov matrix associated to the function f^*

Figure 1: Representations of $f^*(x_1,x_2,x_3) = (x_2 \oplus x_3, x_1 \oplus \overline{x_3}, \overline{x_3})$.

Let us finally present the pseudorandom number generator $\chi_{14Secrypt}$ which is based on random walks in $\Gamma(f)$. More precisely, let be given a Boolean map $f : \mathbb{B}^n \to \mathbb{B}^n$, a PRNG *Random*, an integer b that corresponds to an awaited mixing time, and an initial configuration x^0. Starting from x^0, the algorithm repeats b times a random choice of which edge to follow and traverses this edge. The final configuration is thus outputted. This PRNG is formalized in Algorithm 1 further denoted as $\chi_{14Secrypt}$.

Input: a function f, an iteration number b, an initial configuration x^0 (n bits)
Output: a configuration x (n bits)
$x \leftarrow x^0$;
for $i = 0,\ldots,b-1$ **do**
  $s \leftarrow Random(n)$;
  $x \leftarrow F_f(s,x)$;
end
return x;

Algorithm 1: Pseudo Code of the $\chi_{14Secrypt}$ PRNG

Let $f : \mathbb{B}^n \to \mathbb{B}^n$. It has been shown [1, Th. 4, p. 135] that if its iteration graph is strongly connected, then the output of $\chi_{14Secrypt}$ follows a law that tends to the uniform distribution if and only if its Markov matrix is a doubly stochastic matrix.

The next section presents an efficient method to generate Boolean functions with Doubly Stochastic matrix and Strongly Connected iteration graph, further (abusively) denoted as DSSC matrix.

3 Generation of DSSC Matrices

Finding DSSC matrices can be theoretically handled by Constraint Logic Programming on Finite Domains (CLPFD): all the variables range into finite integer domains with sum and product operations. However, this approach suffers from not being efficient enough for large n due to a *generate and test* pattern.

Intuitivelly, considering the n-cube and removing one outgoing edge and one ongoing edge for each node should be a practical answer to the DSSC matrix finding problem. Moreover, the previous wish of exaclty removing exactly one outgoing and one ongoing edge for each node is solved by removing a Hamiltonian cycle in the n-cube. The next section details this step.

Running example. *The iteration graph of f^* (given in Figure 1(a)) is the* 3*-cube in which the Hamiltonian cycle* $000,100,101,001,011,111,110,010,000$ *has been removed.*

4 Removing Hamiltonian Cycles

The first theoretical section (Section 4.1) shows that this approach produces DSSC matrix, as wished. The motivation to focus on balanced Gray code is then given in Sec. 4.2. We end this section by giving some discussion about practical aspeccts of an existing algorithm that aims at computing such codes (Section 4.3).

4.1 Theoretical Aspects of Removing Hamiltonian Cycles

We first have the following result on stochastic matrix and n-cube without Hamiltonian cycle.

Theorem 1. *The Markov Matrix M resulting from the n-cube in which an Hamiltonian cycle is removed, is doubly stochastic.*

The proof is left as an exercise for the reader. The following result states that the n-cube without one Hamiltonian cycle has the awaited property with regard to the connectivity.

Theorem 2. *The iteration graph issued from the n-cube where an Hamiltonian cycle is removed is strongly connected.*

Again, the proof is left as an exercise for the reader. Removing an Hamiltonian cycle in the n-cube solves thus the DSSC constraint. We are then left to focus on the generation of Hamiltonian cycles in the n-cube. Such a problem is equivalent to find cyclic Gray codes, *i.e.*, to find a sequence of 2^n codewords (n-bits strings) where two successive elements differ in only one bit position and and where the last codeword differs in only one bit position from the first one. The next section is dedicated to these codes.

4.2 Linking to Cyclic (Totally) Balanced Gray Codes

Let n be a given integer. As far as we know, the exact number of Gray codes in $\mathbb{B}^n$ is not known but a lower bound, $\left(\frac{n*\log 2}{e\log\log n}*(1-o(1))\right)^{2^n}$ has been given in [3]. For example, when n is 6, such a number is larger than 10^{13}. To avoid this combinatorial explosion, we want to restrict the generation to any Gray code such that the induced graph of iteration $\Gamma(f)$ is "uniform". In other words, if we count in $\Gamma(f)$ the number of edges that modify the bit i, for $1 \leq i \leq n$, all these values have to be close to each other. Such an approach is equivalent to restrict the search of cyclic Gray codes which are uniform too.

This notion can be formalized as follows. Let $L = w_1, w_2, \ldots, w_{2^n}$ be the sequence of a n-bits cyclic Gray code. Let $S = s_1, s_2, \ldots, s_{2^n}$ be the transition sequence where s_i, $1 \leq i \leq 2^n$ indicates which bit position changes between codewords at index i and $i+1$ modulo 2^n. Let $TC_n : \{1,\ldots,n\} \rightarrow \{0,\ldots,2^n\}$ the *transition count* function that counts the number of times i occurs in S, *i.e.*, the number of times the bit i has been switched in L. The Gray code is *totally balanced* if TC_n is constant (and equal to $\frac{2^n}{n}$). It is *balanced* if for any two bit indices i and j, $|TC_n(i) - TC_n(j)| \leq 2$.

Running example. *Let $L^* = 000, 100, 101, 001, 011, 111, 110, 010$ be the Gray code that corresponds to the Hamiltonian cycle that has been removed in f^*. Its transition sequence is $S = 3,1,3,2,3,1,3,2$ and its transition count function is $TC_3(1) = TC_3(2) = 2$ and $TC_3(3) = 4$. Such a Gray code is balanced.*

Let now $L^4 = 0000, 0010, 0110, 1110, 1111, 0111, 0011, 0001, 0101, 0100, 1100, 1101, 1001, 1011, 1010, 1000$ be a cyclic Gray code. Since $S = 2,3,4,1,4,3,2,3,1,4,1,3,2,1,2,4$, its transition count TC_4 is equal to 4 everywhere and this code is thus totally balanced.

4.3 Induction-Based Generation of Balanced Gray Codes

The article [6] proposed the "Construction B" algorithm to produce Balanced Gray Codes. This method inductively constructs n-bits Gray code given a $n-2$-bit Gray code. The authors have proven that S_n is transition sequence of a cyclic n-bits Gray code if S_{n-2} is. It starts with the transition sequence S_{n-2} of such code and the following first step:

Let l be an even positive integer. Find $u_1, u_2, \ldots, u_{l-2}, v$ (maybe empty) subsequences of S_{n-2} such that S_{n-2} is the concatenation of $s_{i_1}, u_0, s_{i_2}, u_1, s_{i_3}, u_2, ..., s_{i_l-1}, u_{l-2}, s_{i_l}, v$ where $i_1 = 1$, $i_2 = 2$, and $u_0 = \emptyset$ (the empty sequence).

However, this first step is not constructive: it does not precises how to select the subsequences which ensures that yielded Gray code is balanced.

Let us now evaluate the number of subsequences u than can be produced. Since s_{i_1} and s_{i_2} are well defined, we have to chose the $l-2$ elements of $s_3, s_4, \ldots, s_{2^{n-2}}$ that become $s_{i_3}, \ldots, s_{i_l}$. Let $l = 2l'$. There are thus $\#_n = \sum_{l'=1}^{2^{n-3}} \binom{2^{n-2}-2}{2l'-2}$ distinct subsequences u. Numerical values of $\#_n$ are given in table 1. Even for small values of n, it is not reasonable to hope to evaluate the whole set of subsequences.

n	4	5	6	7	8
$\#_n$	1	31	8191	5.3e8	2.3e18
$\#'_n$	1	15	3003	1.4e8	4.5e17

Table 1: Number of distinct u subsequences.

However, it is shown in the article that $TC_n(n-1)$ and $TC_n(n)$ are equal to l. Since this step aims at generating (totally) balanced Gray codes, we have set l to be the largest even integer less or equal than $\frac{2^n}{n}$. This improvement allows to reduce the number of subsequences to study. Examples of such cardinalities are given in Table 1 and are referred as $\#'_n$.

Finally, the table 2 gives the number of non-equivalent functions issued from (totally) balanced Gray codes that can be generated with the approach presented in this article with respect to the number of bits. In other words, it corresponds to the size of the class of generators that can be produced. Notice that when n is 7 and 8, we only give lower bounds for 2.5E5 distinct choices for the u subsequence.

n	4	5	6	7	8
nb. of functions	1	2	1332	> 2300	> 4500

Table 2: Number of Generators w.r.t. the number of bits.

5 Experiments

We have directly implemented the algorithm given in Figure 1. For function f and our experiments b is set with the value given in the fourth column of Table 3.

For each number $n = 4, 5, 6, 7, 8$ of bits, we have generated the functions according the method given in Section 4.3 . For each n, we have then restricted this evaluation to the function whose Markov Matrix has the smallest mixing time. Such functions are given in Table 3. In this table, let us consider for instance the function ⓐ from $\mathbb{B}^4$ to $\mathbb{B}^4$ defined by the following images : $[13, 10, 9, 14, 3, 11, 1, 12, 15, 4, 7, 5, 2, 6, 0, 8]$. In other words, the image of 3 (0011) by ⓐ is 14 (1110): it is obtained as the binary value of the fourth element in the second list (namely 14).

Experiments have shown that all the generators pass the NIST and the DieHARD batteries of tests.

Function f	$f(x)$, for x in $(0,1,2,\ldots,2^n-1)$	n	b
ⓐ	[13,10,9,14,3,11,1,12,15,4,7,5,2,6,0,8]	4	32
ⓑ	[29, 22, 21, 30, 19, 27, 24, 28, 7, 20, 5, 4, 23, 26, 25, 17, 31, 12, 15, 8, 10, 14, 13, 9, 3, 2, 1, 6, 11, 18, 0, 16]	5	41
ⓒ	[55, 60, 45, 56, 43, 62, 61, 40, 53, 50, 52, 36, 59, 34, 57, 49, 15, 14, 47, 46, 11, 58, 33, 44, 7, 54, 39, 37, 51, 2, 32, 48, 63, 26, 25, 30, 19, 27, 17, 28, 31, 20, 23, 21, 18, 22, 16, 24, 13, 12, 29, 8, 10, 42, 41, 0, 5, 38, 4, 6, 35, 3, 9, 1]	6	49
ⓓ	[111,94,93,116,122,114,125,88,87,126,119,84,123,98,81,120,109,78,105,110,99,107,104,108,101, 70,117,96,103,102,113,64,79,30,95,124,83,91,121,24,85,118,69,20,115,90,17,112,77,76,73,12,74, 106,72,8,7,6,71,100,75,82,97,0,127,54,57,62,51,59,56,48,53,38,37,60,55,58,33,49,63,44,47,40,42, 46,45,41,35,34,39,52,43,50,32,36,29,28,61,92,26,18,89,25,19,86,23,4,27,2,16,80,31,10,15,14,3,11, 13,9,5,22,21,68,67,66,65,1]	7	63
ⓔ	[223,250,249,254,187,234,241,252,183,230,229,180,227,178,240,248,237,236,253,172,251,238,201, 224,247,166,165,244, 163,242,161,225,215,220,205,216,218,222,221,208,213,210,135,196,199,132, 194,130,129,200,159,186,185,190,59,170,177,188,191,246,245,52,243,50,176,184,173,46,189,174, 235,42,233,232,231,38,37,228,35,226,33,168,151,156,141,152, 154,158,157,144,149,146,148,150, 155,147,153,145,175,14,143,204,11,202,169,8,7,198,197,4,195,2,1,192,255,124,109,120,107,126, 125,112,103,114,116,100,123,98,121,113,79,106,111,110,75,122,97,108,71,118,117,68,115,66,96, 104,127,90,89,94,83,91,81,92,95,84,87,85,82,86,80,88,77,76,93,72,74,78,105,64,69,102,101,70,99, 67,73,65,55,60,45,56,51,62,61,48,119,182,181,53,179,54,57,49,15,44,47,40,171,58,9,32,167,6,5, 164,3,162,41,160,63,26,25,30,19,27,17,28,31,20,23,21,18,22,16,24,13,10,29,140,43,138,137,12,39, 134,133,36,131,34,0,128]	8	75

Table 3: Functions with DSCC Matrix and smallest MT

6 Conclusion

This article has presented a method to compute a large class of truly chaotic PRNGs. First experiments through the batteries of NIST, and DieHard have shown that the statistical properties are almost established for $n = 4,5,6,7,8$. The iterated map inside the generator is built by removing from a n-cube an Hamiltonian path that corresponds to a (totally) balanced Gray code. The number of balanced gray code is large and each of them can be considered as a key of the PRNG. However, many problems still remain open, most important ones being listed thereafter.

The first one involves the function to iterate. Producing a DSSC matrix is indeed necessary and sufficient but is not linked with the convergence rate to the uniform distribution. To solve this problem, we have proposed to remove from the n-cube an Hamiltonian path that is a (totally) balanced Gray code. We do not have proven that this proposal is the one that minimizes the mixing time. This optimization task is an open problem we plan to study.

Secondly, the approach depends on finding (totally) balanced Gray codes. Even if such codes exist for all even numbers, there is no constructive method to built them when n is large, as far as we know. These two open problems will be investigated in a future work.

References

[1] JacquesM. Bahi, Jean-Francois Couchot, Christophe Guyeux, and Adrien Richard. On the link between strongly connected iteration graphs and chaotic boolean discrete-time dynamical systems. In Olaf Owe, Martin Steffen, and JanArne Telle, editors, *Fundamentals of Computation Theory*, volume 6914 of *Lecture Notes in Computer Science*, pages 126–137. Springer Berlin Heidelberg, 2011.

[2] E. Barker and A. Roginsky. Draft NIST special publication 800-131 recommendation for the transitioning of cryptographic algorithms and key sizes, 2010.

[3] Tomás Feder and Carlos Subi. Nearly tight bounds on the number of hamiltonian circuits of the hypercube and generalizations. *Inf. Process. Lett.*, 109(5):267–272, February 2009.

[4] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. *Markov chains and mixing times*. American Mathematical Society, 2006.

[5] G. Marsaglia. Diehard: a battery of tests of randomness. *http://stat.fsu.edu/ geo/diehard.html*, 1996.

[6] A. J. van Zanten and I. N. Suparta. Totally balanced and exponentially balanced gray codes. *Discrete Analysis and Operational Research*, 11:81–98, 2004.