Pseudorandom Number Generators with Balanced Gray Codes

Keywords: Pseudorandom Number Generator, Theory of Chaos, Markov Matrices, Hamiltonian Paths, Mixing Time, Gray Codes, Statistical Tests

Abstract: In this article, it is shown that a large class of truly chaotic Pseudorandom Number Generators can be constructed. The generators are based on iterating Boolean maps, which are computed using balanced Gray codes. The number of such Gray codes gives the size of the class. 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 formers are called pseudorandom number generators (PRNGs) while the laters 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 (Barker and Roginsky, 2010), DieHARD (Marsaglia, 1996), and TestU01 (Simard and Montréal, 2007), this latter being the most stringent one. 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 (Bahi et al., 2011) 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 this problem can be theoretically solved with a classical constraint logic programming. 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 presented 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 $F_f(i,x)$.

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.*

It is easy to associate a Markov Matrix M to such a graph $G(f)$ as follows:

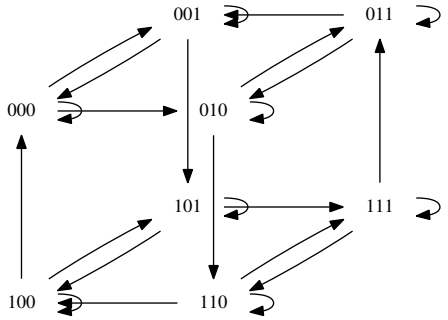

Figure 1: Iteration Graph $\Gamma(f^*)$ of the function f^*

- $M_{ij} = \frac{1}{n}$ if there is an edge from i to j in $\Gamma(f)$ and $i \neq j$;
- $M_{ii} = 1 - \sum_{j=1,j\neq i}^{n} M_{ij}$;
- $M_{ij} = 0$ otherwise.

Running example. *The Markov matrix associated to the function f^* is*

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

It is usual to check whether rows of such kind of matrices converge to a specific distribution. Let us first recall the *Total Variation* distance $\|\pi - \mu\|_{\text{TV}}$, which is defined for two distributions π and μ on the same set Ω by:

$$\|\pi - \mu\|_{\text{TV}} = \max_{A\subset\Omega} |\pi(A) - \mu(A)|.$$

Let then $M(x,\cdot)$ be the distribution induced by the x-th row of M. If the Markov chain induced by M has a stationary distribution π, then we define

$$d(t) = \max_{x\in\Omega} \|M^t(x,\cdot) - \pi\|_{\text{TV}}.$$

Intuitively $d(t)$ is the largest deviation between the distribution π and $M^t(x,\cdot)$, which is the result of iterating t times the function.

Finally, let ε be a positive number, the *mixing time* with respect to ε is given by

$$t_{\text{mix}}(\varepsilon) = \min\{t \mid d(t) \leq \varepsilon\}.$$

It defines the smallest iteration number that is sufficient to obtain a deviation lesser than ε. Notice that the upper and lower bounds of mixing times cannot

directly be computed with eigenvalues formulae as expressed in (Levin et al., 2006, Chap. 12). The authors of this latter work only consider reversible Markov matrices whereas we do no restrict our matrices to such a form.

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.

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

This PRNG is a particularized version of Algorithm given in (Bahi et al., 2011). Compared to this latter, the length of the random walk of our algorithm is always constant (and is equal to b) whereas it was given by a second PRNG in this latter. However, all the theoretical results that are given in (Bahi et al., 2011) remain true since the proofs do not rely on this fact. Let us recall the following theorem.

Theorem 1 ((Bahi et al., 2011, Th. 4, p. 135))**.** *Let $f : \mathbb{B}^n \to \mathbb{B}^n$, $\Gamma(f)$ its iteration graph, and M its Markov matrix . If $\Gamma(f)$ is strongly connected, then the output of the PRNG follows a law that tends to the uniform distribution if and only if M is a doubly stochastic matrix.*

With all this material, we can present 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

This aim of this section is to show that finding DSSC matrices from a hypercube is a typical finite domain satisfaction problem, classically denoted as Constraint Logic Programming on Finite Domains (CLPFD). This part is addressed in the first section. Next, we analyse the first results to provide a generation of DSSC matrices with small mixing times.

3.1 Constraint Logic Programming on Finite Domains

First of all, let n be the number of elements. In order to avoid fractions in this article, we consider here that the sum of each column and each row is n. It can easily be normalized to 1. The goal is thus to find all the $2^n \times 2^n$ matrices M such that:

1. M_{ij} is 0 if j is not a neighbor of i, *i.e.*, there is no edge from i to j in the n-cube.
2. $0 \leq M_{ii} \leq n$: the number of loops around i is lesser than n
3. Otherwise $0 \leq M_{ij} \leq 1$: if the edge from i to j is kept, M_{ij} is 1, and 0 otherwise.
4. For any index of line i, $1 \leq i \leq 2^n$, $n = \sum_{1 \leq j \leq 2^n} M_{ij}$: the matrix is right stochastic.
5. For any index of column j, $1 \leq j \leq 2^n$, $n = \sum_{1 \leq i \leq 2^n} M_{ij}$: the matrix is left stochastic.
6. All the values of $\sum_{1 \leq k \leq 2^n} M^k$ are strictly positive, *i.e.*, the induced graph is strongly connected.

Since these variables range into finite integer domains with sum and product operations, this problem can be theoretically handled by Constraint Logic Programming on Finite Domains (CLPFD), as implemented in prolog. The algorithm given in Figure 2 is indeed the core of the prolog file that is used to instantiate all the solutions when n is 2. In this code, `mmult(X,Y,R)` and `summ(X,Y,R)` are true if and only if R is the matrix product or the matrix sum between X and Y respectively. It is not hard to adapt such a code to any value of positive integer n.

Finally, we define the relation $\mathcal{R}$, which is established on the two functions f and g if the iteration graphs $\Gamma(f)$ and $\Gamma(g)$ are isomorphic. Obviously, this is an equivalence relation.

3.2 Analysis of the Approach

When executed on a personal computer, prolog finds in less than 1 second the 49 solutions when n is 2, where only 2 are not equivalent, and in less than 1 minute the 27642 solutions where only 111 are not equivalent when n is 3. But it does not achieve the generation of all the solutions when n is 4. This approach suffers from not being efficient enough for large n due to a *generate and test* approach, despite the efficiency of the native backtrack of in CLP.

```
bistoc(X):-
 M=[[M0_0, M0_1, 0, M0_3], [M1_0, M1_1, 0, M1_3],
    [M2_0, 0, M2_2, M2_3], [0, M3_1, M3_2, M3_3]],
 [M0_0, M1_1, M2_2, M3_3] ins 0..2,
 [M0_1, M0_3, M1_0, M1_3, M2_0, M2_3, M3_1, M3_2]
    ins 0..1,
 M0_0+ M0_1+ M0_2 #=2, M1_0+ M1_1+ M1_3 #=2,
 M2_0+ M2_2+ M2_3 #=2, M3_1+ M3_2+ M3_3 #=2,
 M0_0+ M1_0+ M2_0 #=2, M0_1+ M1_1+ M3_1 #=2,
 M0_2+ M2_2+ M3_2 #=2, M1_3+ M2_3+ M3_3 #=2,
 mmult(M,M,M2),
 mmult(M,M2,M3),
 mmult(M,M3,M4),
 summ(M,M2,S2),
 summ(S2,M3,S3),
 summ(S3,M4,S4),
 allpositive(S4).
```

Figure 2: Prolog Problem to Find DSSC Matrix when $n = 2$

However, first results for small values of n have been evaluated. More precisely, non equivalent generated functions have been compared according to their ability to efficiently produce uniformly distributed outputs, *i.e.*, to have the smallest mixing time.

Running example. *Table 1 gives the 5 best Boolean functions ordered by their mixing times (MT) in the third column for* $\varepsilon = 10^{-5}$.

Name	Image	MT
f^*	$(x_2 \oplus x_3, x_1 \oplus \overline{x_3}, \overline{x_3})$	16
f^a	$(x_2 \oplus x_3, \overline{x_1}x_3 + x_1\overline{x_2}, \overline{x_1}x_3 + x_1x_2)$	17
f^b	$(\overline{x_1}(x_2+x_3)+x_2x_3, \overline{x_1}(\overline{x_2}+\overline{x_3})+\overline{x_2}x_3, \overline{x_3}(\overline{x_1}+x_2)+\overline{x_1}x_2)$	26
f^c	$(\overline{x_1}(x_2+x_3)+x_2x_3, \overline{x_1}(\overline{x_2}+\overline{x_3})+\overline{x_2}x_3, \overline{x_3}(\overline{x_1}+x_2)+\overline{x_1}x_2)$	29
f^d	$(x_1 \oplus x_2, x_3(\overline{x_1}+\overline{x_2}), \overline{x_3})$	30

Table 1: The 5 Boolean functions with smallest MT when $n = 3$.

A syntactical analysis of the functions for n=3 does not help to understand how to build a Boolean map with a small mixing time. However the iteration graph of f^* (given in Figure 1) is the 3-cube in which the cycle $000, 100, 101, 001, 011, 111, 110, 010, 000$ has been removed. This cycle that visits each vertex exactly once is usually denotes as *Hamiltonian cycle*. We are now focusing on the generation of DSSC matrices by removing Hamiltonian cycles of the n-cube. This is the aims of the next section.

4 REMOVING HAMILTONIAN CYCLES

This section addresses the problem of removing an Hamiltonian path in the n-cube. The first theoretical section 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 recalling an 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 2. *The Markov Matrix M resulting from the n-cube in which an Hamiltonian cycle is removed, is doubly stochastic.*

Proof. An Hamiltonian cycle visits each vertex exactly once. For each vertex v in the n-cube, one ongoing edge (o,v) and one outgoing edge (v,e) are thus removed.

Let us consider the Markov matrix M of the n-cube. It is obviously doubly stochastic. Since we exactly remove one outgoing edge, the value of M_{ve} decreases from $\frac{1}{n}$ to 0 and M_{vv} is $\frac{1}{n}$. The M matrix is stochastic again. Similarly for ongoing edge, since one ongoing edge is dropped for each vertex, the value of M_{ov} decreases from $\frac{1}{n}$ to 0. Moreover, since M_{vv} is $\frac{1}{n}$, the sum of values in column v is 1, and M is doubly stochastic. □

The following result states that the n-cube without one Hamiltonian cycle has the awaited property with regard to the connectivity.

Theorem 3. *The iteration graph issued from the n-cube where an Hamiltonian cycle is removed is strongly connected.*

Proof. We consider the reverse cycle r of the Hamiltonian cycle c. There is no edge that belongs to both r and c: otherwise c would contain one vertex twice. Thus, no edges of r has been removed. The cycle r is obviously an Hamiltonian cycle and contains all the nodes. Any node of the n-cube where c has been removed can thus reach any node. The iteration graph is thus strongly connected. □

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 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. The next section is dedicated to these codes.

4.2 Linking to (Totally) Balanced Gray Codes

Many research works (Bhat and Savage, 1996, Zanten and Suparta, 2004, Flahive and Bose, 2007) have addressed the subject of finding Gray codes. Since our approach is based on removing a cycle, we are concerned with cyclic Gray codes, *i.e.*, sequences where the last codeword differs in only one bit position from the first one.

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 (Feder and Subi, 2009). 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* $S = 2,3,4,1,4,3,2,3,1,4,1,3,2,1,2,4$ TC_4 *is equal to 4 everywhere, this code is thus totally balanced.*

On the contrary, for the standard 4-bits Gray code $L^{st} = 0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000$, *we have* $TC_4(1) = 8$ $TC_4(2) = 4$ $TC_4(3) = TC_4(4) = 2$ *and the code is neither balanced nor totally balanced.*

4.3 Induction-Based Generation of Balanced Gray Codes

The algorithm we adapt is based on the "Construction B" (Zanten and Suparta, 2004) we recall now. This method inductively constructs n-bits Gray code given a $n-2$-bit Gray code.

It starts with the transition sequence S_{n-2} of such code.

1. 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).
2. Replace in S_{n-2} the sequences $u_0, u_1, u_2, \ldots, u_{l-2}$ by $n-1, u'(u_1, n-1, n), u'(u_2, n, n-1), u'(u_3, n-1, n), \ldots, u'(u_{l-2}, n, n-1)$ respectively, where $u'(u,x,y)$ is the sequence u, x, u^R, y, u such that u^R is u in reversed order. The obtained sequence is further denoted as U.
3. Construct the sequences $V = v^R, n, v$, $W = n-1, S_{n-2}, n$, and let W' be W where the first two elements have been exchanged.
4. The transition sequence S_n is thus the concatenation U^R, V, W'.

It has been proven in (Zanten and Suparta, 2004) that S_n is transition sequence of a cyclic n-bits Gray code if S_{n-2} is. However, the step 1 is not a constructive step that 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 2. Even for small values of n, it is not

n	5	6	7	8	9
$\#_n$	31	8191	5.3e8	2.3e18	4.2e37
$\#'_n$	15	3003	1.4e8	4.5e17	1.6e36

Table 2: Number of distinct u subsequences.

reasonable to hope to evaluate the whole set of 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 2 and are refered as $\#'_n$.

Finally, the table 3 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.

n	3	4	5	6
nb. of functions	1	1	2	1332

Table 3: Number of Generators w.r.t. the number of bits.

5 EXPERIMENTS

We present in Algorithm 2 the method that allows to take any chaotic function as the core of a pseudo random number generator. Among the parameters, it takes the number b of minimal iterations that have to be executed to get a uniform like distribution. For function f and our experiments b is set with the value given in the fourth column of Table 4.

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 mod n*;
  $x \leftarrow (x - (x\&(1 << s)) + f(x)\&(1 << s))$;
end
return x;

Algorithm 2: Pseudo Code of the $\chi_{14Secrypt}$ PRNG

For each number $n = 4, 6, 8$ of bits, we have generated all the functions according the method given in Section 4. 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 4. 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).

5.1 NIST

In our experiments, 100 sequences (s = 100) of 1,000,000 bits are generated and tested. If the value $\mathbb{P}_T$ of any test is smaller than 0.0001, the sequences are considered to be not good enough and the generator is unsuitable. Table 5 shows $\mathbb{P}_T$ of sequences based on discrete chaotic iterations using different schemes. If there are at least two statistical values in a test, this test is marked with an asterisk and the average value is computed to characterize the statistics. We can see in Table 5 that all the rates are greater than 97/100, *i. e.*, all the generators pass the NIST test.

5.2 DieHARD

Table 6 gives the results derived from applying the DieHARD battery (Marsaglia, 1996) of tests to the PRNGs considered in this work. As it can be observed, all the generator presented in this document can pass the DieHARD battery of tests.

5.3 TestU01

TestU01 (Simard and Montréal, 2007) is a software library that provides general implementations of the classical statistical tests for random number generators, as well as several others proposed in the literature, and some original ones. This library is currently the most reputed and stringent one for testing the randomness profile of a given sequence. TestU01 encompasses the NIST and DieHARD tests suites with 2 batteries specific to hardware based generators (namely, Rabbit and Alphabit). Its three core batteries of tests are however SmallCrush, Crush, and BigCrush, classified according to their difficulty.

To date, we can claim after experiments that ⓐ generator is able to pass the 15 tests embedded into the SmallCrush battery and it succedded too to pass the 144 tests of the Crush one. BigCrush results on ⓐ are expected soon, while TestU01 has been launched too on generators having the other iteration functions detailed in this article.

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
ⓑ	[55, 60, 45, 56, 58, 62, 61, 40, 53, 38, 52, 54, 35, 51, 33, 49, 39, 14, 47, 46, 59, 43, 57, 44, 37, 6, 36, 4, 3, 50, 1, 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, 15, 2, 7, 5, 11, 34, 9, 32]	6	49
ⓒ	[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 4: Functions with DSCC Matrix and smallest MT

Table 5: NIST SP 800-22 test results ($\mathbb{P}_T$)

Method	ⓐ	ⓑ	ⓒ
Frequency (Monobit)	0.678 (0.99)	0.574 (0.99)	0.699 (0.96)
Frequency within a Block	0.102 (0.99)	0.816 (0.98)	0.419 (0.99)
Runs	0.171 (0.98)	0.657 (0.98)	0.554 (0.99)
Longest Run of Ones in a Block	0.115 (0.98)	0.534 (1.0)	0.534 (0.98)
Binary Matrix Rank	0.401 (0.97)	0.554 (0.99)	0.911 (0.99)
Discrete Fourier Transform (Spectral)	0.554 (0.98)	0.350 (0.98)	0.080 (0.99)
Non-overlapping Template Matching*	0.509 (0.990)	0.443 (0.990)	0.499 (0.989)
Overlapping Template Matching	0.437 (0.99)	0.699 (1.0)	0.236 (0.99)
Maurer's "Universal Statistical"	0.171 (0.99)	0.000 (0.97)	0.657 (0.99)
Linear Complexity	0.171 (1.0)	0.637 (0.99)	0.834 (1.0)
Serial* (m=10)	0.435 (1.0)	0.565 (0.98)	0.592 (1.0)
Approximate Entropy (m=10)	0.137 (0.98)	0.867 (0.99)	0.062 (1.0)
Cumulative Sums (Cusum) *	0.580 (0.99)	0.368 (0.99)	0.569 (0.97)
Random Excursions *	0.245 (0.980)	0.421 (0.991)	0.656 (0.985)
Random Excursions Variant *	0.292 (0.991)	0.450 (0.990)	0.520 (0.996)
Success	15/15	15/15	15/15

Table 6: Results of DieHARD battery of tests

No.	Test name	Generators		
		ⓐ	ⓑ	ⓒ
1	Overlapping Sum	Pass	Pass	Pass
2	Runs Up 1	Pass	Pass	Pass
	Runs Down 1	Pass	Pass	Pass
	Runs Up 2	Pass	Pass	Pass
	Runs Down 2	Pass	Pass	Pass
3	3D Spheres	Pass	Pass	Pass
4	Parking Lot	Pass	Pass	Pass
5	Birthday Spacing	Pass	Pass	Pass
6	Count the ones 1	Pass	Pass	Pass
7	Binary Rank 6×8	Pass	Pass	Pass
8	Binary Rank 31×31	Pass	Pass	Pass
9	Binary Rank 32×32	Pass	Pass	Pass
10	Count the ones 2	Pass	Pass	Pass
11	Bit Stream	Pass	Pass	Pass
12	Craps Wins	Pass	Pass	Pass
	Throws	Pass	Pass	Pass
13	Minimum Distance	Pass	Pass	Pass
14	Overlapping Perm.	Pass	Pass	Pass
15	Squeeze	Pass	Pass	Pass
16	OPSO	Pass	Pass	Pass
17	OQSO	Pass	Pass	Pass
18	DNA	Pass	Pass	Pass
	Number of tests passed	18	18	18

6 CONCLUSION

This article has presented a method to compute a large class of truly chaotic PRNGs. First experiments through the batteries of NIST, DieHard, and TestU01 have shown that the statistical properties are almost established for $n = 4, 6, 8$. 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

Bahi, J., Couchot, J.-F., Guyeux, C., and Richard, A. (2011). On the link between strongly connected iteration graphs and chaotic boolean discrete-time dynamical systems. In Owe, O., Steffen, M., and Telle, J., editors, *Fundamentals of Computation Theory*, volume 6914 of *Lecture Notes in Computer Science*, pages 126–137. Springer Berlin Heidelberg.

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

Bhat, G. S. and Savage, C. D. (1996). Balanced gray codes. *Electr. J. Comb.*, 3(1).

Feder, T. and Subi, C. (2009). Nearly tight bounds on the number of hamiltonian circuits of the hypercube and generalizations. *Inf. Process. Lett.*, 109(5):267–272.

Flahive, M. and Bose, B. (2007). Balancing cyclic r-ary gray codes. *Electr. J. Comb.*, 14(1).

Levin, D. A., Peres, Y., and Wilmer, E. L. (2006). *Markov chains and mixing times*. American Mathematical Society.

Marsaglia, G. (1996). DieHARD: a battery of tests of randomness. *http://stat.fsu.edu/ geo/diehard.html*.

Simard, R. and Montréal, U. D. (2007). Testu01: A c library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, page 2007.

Zanten, A. J. v. and Suparta, I. N. (2004). Totally balanced and exponentially balanced gray codes. *Discrete Analysis and Operational Research*, 11:81–98.