Gray Codes Generation Algorithm and Theoretical Evaluation of Random Walks in N-Cubes

Sylvain Contassot-Vivier, Jean-François Couchot, Pierre-Cyrille Héam

Gray Codes Generation Algorithm and Theoretical Evaluation of Random Walks in N-Cubes

Sylvain Contassot-Vivier [1], Jean-François Couchot [2] and Pierre-Cyrille Héam [2]

[1] Université de Lorraine, CNRS, LORIA, F-54000 Nancy, France
Sylvain.Contassotvivier@loria.fr
[2] Université Bourgogne Franche-Comté, FEMTO-ST Institute, F-90000 Belfort, France
Jean-Francois.Couchot@univ-fcomte.fr, Pierre-Cyrille.Heam@femto-st.fr

Abstract

In previous works, some of the authors have proposed a canonical form of Gray Codes (GCs) in N-cubes (hypercubes of dimension N). This form allowed them to draw an algorithm that theoretically provides exactly all the GCs for a given dimension N. In another work, we first have shown that any of these GC can be used to build the transition function of a Pseudorandom Number Generator (PRNG). Also, we have found a theoretical quadratic upper bound of the mixing time, i.e., the number of iterations that are required to provide a PRNG whose output is uniform. This article, extends these two previous works both practically and theoretically. On the one hand, another algorithm for generating GCs is proposed that provides an efficient generation of subsets of the entire set of GCs related to a given dimension N. This offers a large choice of GC to be used in the construction of Choatic Iterations based PRNGs (CI-PRNGs), leading to a large class of possible PRNGs. On the other hand, the mixing time has been theoretically shown to be in $\mathsf{N}\log(\mathsf{N})$, which was anticipated in the previous article, but not proven.

Keywords: Gray Codes; Hamiltonian cycles; N-cube; mixing time; PRNG

1 Introduction

Random walk in the N-cube is the process of randomly crossing edges in the hypercube of N dimensions. This theoretical subject has received a lot of attention in recent years [19, 13, 15, 24, 18]. The article [11] is situated in this thematic field. Indeed, it focuses on a Random Walk in a N-cube where a balanced Hamiltonian cycle has been removed. It is worth recalling that a Hamiltonian cycle in the N-cube is also a Gray Code (GC). To compute such kind of cycle, an algebraic solution of an undeterministic approach [25] is presented. Solving the induced system of linear equations is an easy and practicable task. However, the number of solutions found by this method is infinitely smaller than what generally exists if the approach [25] was not used.

In another work [10], we have proposed a canonical form of GCs in N-cubes. This form allows to draw an algorithm, based on Wild's scheme [27], that theoretically provides exactly all the GCs for a given dimension N. However, due to the exponential growth of the number of GCs with the dimension of the N-cube, the number of produced GCs must be specified as a boundary argument. Moreover, we have observed that the time to generate a first GC may be quite large when the dimension of the N-cube increases.

Once a GC is produced, it can be used to build a random walk in the N-cube based post-processing of the output of any classical PRNG [12, 11, 4]. This post-processing has many advantages: compared to the original PRNG which does not have a chaotic behavior according to Devaney, it first adds this property [1]. It moreover preserves the cryptographical security of the embedded PRNG [14], which may be required in a cryptographic context. Finally, it largely improves the statistical properties of the on board PRNG [3] leading to a PRNG that successfully passes not only the classical statistical tests like [21], but also the most difficult ones, namely the TestU01 [17]. In particular, it has been already presented in ([2] Chap. 5, p. 84+) that such post-processing provides a new PRNG at minimal cost which is 2.5 times faster than the PRNGs which pass the whole TestU01 [17]. Moreover, it is worth noticing that most of the classical PRNGs [20, 16] do not pass the TestU01.

In such a context, the mixing time [18] corresponds to the number of steps that are required to reach any vertex with the same probability according to a given approximation tolerance ε. In other words, it is the time until the set of vertices is ε-close to the uniform distribution. Such a theoretical model provides information on the statistical quality of the obtained PRNG. The lower the mixing time, the more uniform the PRNG output.

We first have shown that any of the generated GC can be used to build the transition function of a Pseudorandom Number Generator (PRNG) based on random walk in the N-cube. Then, we had found a theoretical quadratic upper bound of the mixing time.

This article is twofold as it extends these two previous works both practically and theoretically. In the first part (Section 2), we present another algorithm that can generate many GCs in a given N-cube more efficiently than our previous algorithm. Moreover, filtering rules can be applied during the generation in order to choose codes that are especially adapted to the use in PRNG based on random walks. This leads to a very large set of possible PRNGs. In the second part (Section 4), the mixing time has been theoretically shown to be in $\mathsf{N}.\log_2(\mathsf{N})$, which was anticipated, but not yet proven. Then, experimental results are given and discussed in Section 5.

2 Gray Codes Generation Algorithms

In this part, we propose another algorithm that efficiently generates GCs in a given N-cube, starting from a list of known GCs. The initial list may be reduced to a singleton.

2.1 Inverting Algorithm

Many approaches to generate GCs in a N-cube start from one or more known codes in dimensions smaller than N [23, 6, 25, 26, 8]. However, such methods present either strong limitations over the type of cycles that can be generated, or large amount of computations that lead to non efficient generation algorithms.

The approach we propose in this paper is quite different as it builds new Gray codes that have the same dimension as the initial known code. The principle consists in transforming one GC into another non-isomorphic GC by an adequate operation. In the sequel, we call *neighbors*, two GCs that are directly linked by such an operation. Hence, the key idea of the approach is to explore the space of possible GCs by generating the neighbors of the initial code and then re-applying the same strategy to the obtained codes. Therefore, in a sense, we perform a transitive closure according to the considered code transformation as depicted in Figure 1.

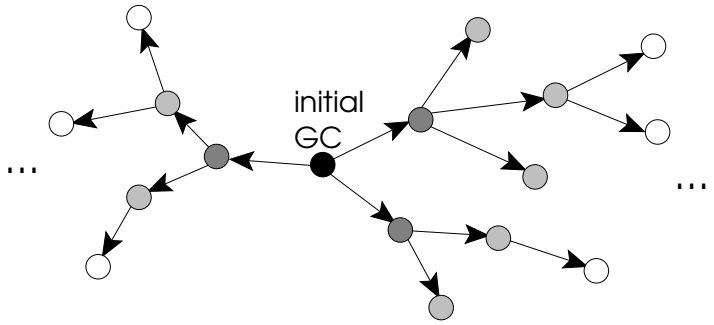


Figure 1: Transitive exploration of the space of Gray Codes (GCs) from the initial black code. Each dot represents one GC. The color gets lighter at each additional transform.

2.2 GC Transformation Principle

The core issue in the proposed approach above is to find a code transformation that generates another valid Gray code that is not isomorphic to the initial code. In this context, we propose what we call the *inversion transformation*. It consists in finding a particular sub-sequence of the initial code that can be inverted to produce another valid Gray code. If C is a GC in dimension N whose sequence of vertices is $(C_0, ..., C_{2^N-1})$, the problem is to find a sub-sequence $(C_i, ..., C_j)$ in C such that the code $(C_0, ..., C_{i-1}, C_j, C_{j-1}, ..., C_{i+1}, C_i, C_{j+1}, ..., C_{2^N-1})$ is also a valid GC.

We have identified conditions over the four vertices C_{i-1}, C_i, C_j and C_{j+1} that fulfill that constraint:

1. the four vertices must be on the same face of the N-cube, i.e., the differences between the four vertices are contained in only two dimensions of the N-cube
2. C_i and C_j must be diagonally opposed inside the face and, as a consequence, C_{i-1} and C_{j+1} are diagonally opposed too (this is equivalent to say that transitions (C_{i-1}, C_i) and (C_j, C_{j+1}) go in the same direction along their dimension)

Figure 2 illustrates the configuration providing a valid sub-sequence inversion.

Those conditions comes from the fact that in order to obtain a valid GC, the exchanges of C_i and C_j in the sequence must satisfies the condition of Gray codes. This implies that the number of

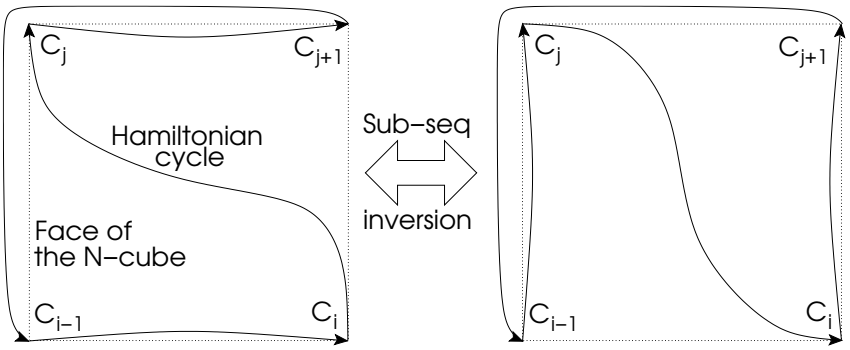


Figure 2: Inversion of sub-sequence $(C_i, ..., C_j)$ inside a face of the N-cube (dotted square).

differences between C_{i-1} and C_j must be only one, and this is the same for C_i and C_{j+1}. Therefore, as C_i and C_j are at a Hamming distance of 1 from C_{i-1} and they are different vertices (by construction of the initial GC), they must be exactly at a Hamming distance of two to each other. So, they are diagonally opposed on a same face of the N-cube and C_{i-1} is also on that face (at one of the other two corners). The same reasoning applies for C_i, C_j and C_{j+1}, leading to the conclusion that C_{j+1} must be at the last free corner of the same face. Therefore, it is diagonally opposed to C_{i-1}, leading to the configuration in Figure 2.

By construction, it is sure that the obtained cycle is a valid Hamiltonian cycle (Gray code in the N-cube). Indeed, as the inversion operation is equivalent to change only some transitions of the input code, the output code contains exactly the same set of vertices as the input one. Therefore, the generated code satisfies the property of Hamiltonian cycle as it contains the total number of vertices of the considered graph. Moreover, as the Gray code property is independent of the cycle direction, the inverted sub-sequence $(C_j, ..., C_i)$ is still a valid part of a Gray code. Also, as explained before, the two conditions enumerated above ensure that both transitions (C_{i-1}, C_j) and (C_i, C_{j+1}), connecting the inverted sub-sequence and the unchanged part of the initial code, take place along one dimension of the N-cube, thus satisfying the Gray code property. Therefore the generated code fulfills all the constraints of Gray codes in the N-cube. An example of code generation in dimension 4 is given in Figure 3.

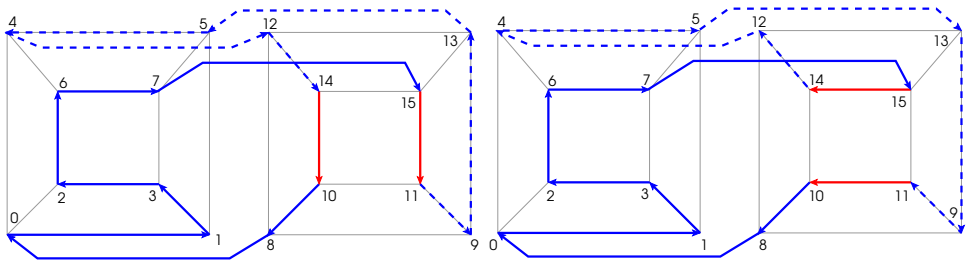


Figure 3: Inversion of sub-sequence $(11, ..., 14)$ (dotted line) on face (10, 11, 14, 15) in the 4-cube.

A first consequence of the sub-sequence inversion is that the codes generated from one initial code have different dimension frequencies than their source. We remind the reader that the dimension frequencies of a GC are the numbers of times each dimension is traversed by the code. The mentioned consequence comes from the fact that when inverting the sequence $(C_i, ..., C_j)$ in the initial code, the transitions (C_{i-1}, C_i) and (C_j, C_{j+1}) that are along the same dimension (opposite edges of the face), are respectively replaced by the transitions (C_{i-1}, C_j) and (C_i, C_{j+1}), that are along the other dimension of the face. This fact, that can be observed in Figure 2, implies that between the two cycles, the frequency of one dimension decreases by two while the frequency of another one increases by two. This is important when wanting to guide the generation towards cycles with specific properties, like the global balance of frequencies.

Another consequence is that the sub-sequence $(C_i, ..., C_j)$ must contain at least five vertices to go from C_i to C_j by traversing one dimension at a time while avoiding the vertices C_{i-1} and C_{j+1}. This reasoning applies also to the sub-sequence $(C_{j+1}, ..., C_{i-1})$ for the same reason. Such information can be exploited to optimize the generation algorithm.

2.3 Generation Algorithm from a Known GC

Algorithm 1 presents the sketch of the GC generation algorithm by sub-sequence inversion.

For each position in the initial sequence C, the algorithm takes the transition from this vertex as a reference and searches for any *parallel transition* in the sequence. By parallel transition, we mean a transition along the same dimension and in the same direction, whose vertices form a face of the N-cube together with the vertices of the reference transition.

In this algorithm, function `dimOfTransition(x,y)` returns the dimension index corresponding to the transition formed by the two neighbors `x` and `y` in the N-cube (complexity $\mathcal{O}(1)$). Also, function `bitInversion(x,i)` inverses the bit `i` of vector `x` ($\mathcal{O}(1)$). Function `copyAndInvertSubSeq(C,a,b)` returns a copy of cycle `C` in which the sub-sequence between indices `a` and `b` is inverted ($\mathcal{O}(|C|)$). Finally, function `exclusiveInsert(C,L)` inserts cycle `C` in list `L` only if `C` is not yet in `L`, and it returns the Boolean `true` when the insertion is performed and `false` otherwise ($\mathcal{O}(\log_2(|L|))$ thanks to the total order induced by the canonical form).

As depicted in Figure 4a, there are $N-1$ possible parallel transitions for a given transition in the N-cube. The dotted lines exhibit the dimensions that must be parsed to get the other neighbors of the reference transition source, which are not the reference transition destination. In this particular example, the algorithm would search for sub-sequences of the form $(1,...,2,3)$, $(1,...,4,5)$ and $(1,...,8,9)$. Each time one of these sub-sequences is found, a new GC can be produced. It is worth noticing that several sub-sequences may be valid for one reference transition, as shown in Figure 4b. In that example, the red transitions are valid parallel transitions of the black transition. Therefore, two distinct cycles can be generated by inverting either $(1,...,2)$ or $(1,...,4)$.

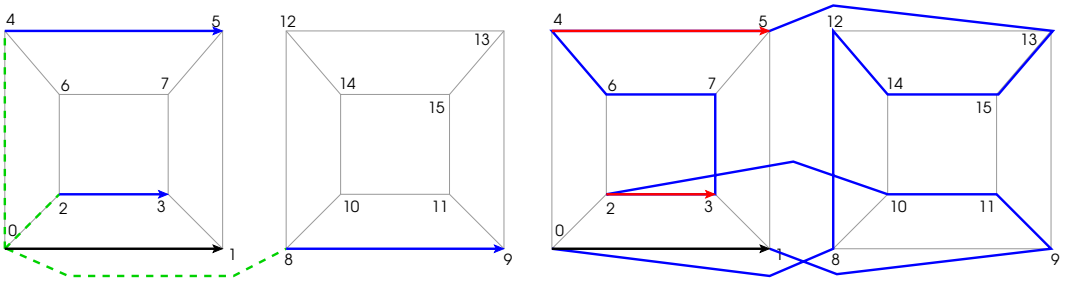
Algorithm 1: Function generateFrom(C).

```
Input: a Gray code C of dimension n
Output: a list lst of Gray codes deduced from C

V ← 2^n                                  // Number of vertices in C
for i from 0 to V−1 do                   // Parsing of sequence C
  d ← dimOfTransition(C_i, C_{(i+1)%V})  // Dimension used by the transition
  for j from 0 to n−1 do                 // Parsing of dimensions of the N − cube
    if j ≠ d then                        // For any other dimension than d
      src ← bitInversion(C_i, j)         // Neighbor of C_i along dimension j
      dst ← bitInversion(src, d)         // Neighbor of src along dimension d
      // vertices C_i, C_{i+1}, src and dst form a face of the N − cube
      // We look for the presence of transition src → dst in C
      found ← false                      // Transition src → dst not yet found in C
      k ← 5                              // Starting search offset from i
      while k ≤ V−5 and ¬found do        // Search of transition src → dst in C
        p ← i + k                        // probing transition src → dst at offset k from i
        if C_{p%V} = src and C_{(p+1)%V} = dst then // transition found!
          found ← true
          // Bulding of a new cycle from C with sub−seq (C_{i+1},...,C_p) inverted
          newCycle ← copyAndInvertSubSeq(C, i+1, p)
          // Insertion of the new cycle in lst only if not yet present
          exclusiveInsert(newCycle, lst)
        endif
        k ← k + 1
      endwhile
    endif
  endfor
endfor
return lst
```

To be exhaustive, and to get all the possible inversions, the algorithm applies the search for parallel transitions successively to all the transitions in the initial sequence. Also, the constraint over the minimal length of the sub-sequence to be inverted and of the unchanged part of the cycle, mentioned in the previous section, has been taken into account in the inner part of our algorithm, when searching the considered parallel transition in the cycle (boundaries of variable k). Although this does not change the overall complexity of the algorithm, it saves computations and slightly reduces the execution time. For clarity sake, the algorithm is given in its simplest form, but additional features could be added, such as a filtering over the generated cycles according to some particular properties, or a more subtle selection and guiding of the sub-sequences searches that would avoid unnecessary computations when looking for specific GCs. Also, sorting the list lst according to the canonical forms of the generated cycles is recommended to perform faster exclusions or insertions.

Finally, we obtain an algorithm whose complexity to generate all the cycles that can be derived from one GC of length V is $\mathcal{O}\left(V^2.\log_2(V)\right)$.This corresponds to the complexity $\mathcal{O}\left(N.4^N\right)$ for the corresponding N-cube of dimension N. Also, the memory complexity of the algorithm is $\mathcal{O}(c.V)$, where c is the number of generated cycles ($c = |lst|$). This algorithm alone is not sufficient to generate as many cycles as wanted. In fact, it is trivial to deduce a coarse upper bound of the number of



(a) Possible parallel transitions (blue) for a reference transition (black).

(b) Two valid parallel transitions (red) for the reference transition (black) in the blue cycle.

Figure 4: Examples of possible and valid parallel transitions in the 4-cube.

possible generations from one GC of length V, which is $V.(\log_2(V) - 1)$. Thus, as explained in Section 2.1, we have designed a transitive closure algorithm that uses GC generation algorithm.

2.4 Transitive Closure Algorithm

Our transitive closure algorithm is an iterative algorithm that begins with a given set of known GCs (at least one). At the first step, each GC in the initial set is used as input to the GC generation algorithm described before. Then, the list of input GCs for the next step is rebuilt by including in it only the new cycles that have not yet been used as input GC. This clarification is important because one GC may be generated several times from different sources. So, it is necessary to maintain a list of all the GCs that have already been treated since the beginning of the process, in order to avoid duplicate work. At the next step, the GCs in the built list are used as input to the GC generation algorithm, and so on until no more GC is generated or a specific number of GCs has been reached. The overall process is detailed in Algorithm 2, in which the function `generateFrom()` is the one detailed in Algorithm 1.

Algorithm 2: Transitive closure generation algorithm.

```
1  Input: a set I of initial Gray codes of dimension n
2  Output: a list lst of Gray codes obtained from I by transitive closure

4  lst ← I          // Initial GCs are put in the list to avoid re-process
5  il ← I           // Input list for the first step
6  finished ← false // Process not finished at the beginning
7  while ¬finished do          // Main loop of the transitive closure
8    nl ← ∅                    // Init of the list of GCs created for the 1st time
9    foreach i in il do        // Processing of every GC i in the input list
10     gen ← generateFrom(i)   // Generation of new GCs from source i
11     foreach j in gen do     // Checking for insertion of each generated GC:
12       if exclusiveInsert(j, lst) then  // in lst if first creation (not yet in it)
13         insert(j, nl)                  // and also in nl in this case (next step)
14       endif
15     endforeach
16   endforeach
17   il ← nl                   // New GCs at this step become input GCs of next step
18   finished ← decision(lst) // Stopping decision according to stopping criteria
19 endwhile
20 return lst
```

Concerning the stopping criteria of the iterative process, it must include the detection that no new code has been produced at the last iteration. Also, as soon as the dimension of the N-cube is greater than 5, the number of potential generations becomes very large and the stopping criteria must include a limitation of the number of generations. As for the previous algorithm, the lists lst and nl must be sorted according to the canonical forms in order to obtain fast searches and insertions.

The complexity of that algorithm is deduced by analyzing the nested loops. Although the number of iterations of the global `while` loop is variable, we can compute the total number of iterations of inner loops on `i` (line 9) and on `j` (line 11). This allows us to deduce an upper bound of the global complexity as the other actions performed in the `while` loop: the initialization of `nl`, the copy of `nl` into `il` and the stopping decision, take either constant or linear times (naive copy of `il` into `nl`, but optimization is possible) that are negligible compared to the total cost of the loop on `i`.

Concerning the loop on `i`, as the `il` list stores only the new cycles at each iteration of the closure process, we can deduce that each generated cycle is stored only once in this list, and thus is processed only once by the loop. Thus, the total number of iterations performed by this loop is equal to the total number of generated cycles (denoted by L). Inside this loop, the cost of `generateFrom` is $N.4^N$

as seen in the previous sub-section. The number of cycles generated by `generateFrom` may vary from one cycle `i` to another. However, an amortized complexity analysis using α as the average number of generations per cycle allows us to deduce that the total number of iterations performed by the inner loop in `j` is $\alpha.L$. This last loop is dominated by the exclusive insertions in the sorted list `lst`. Here also, the size of that list increases during the whole process, from 0 to L elements. In order to compute the total cost of the successive calls to `exclusiveInsert()`, we define S as the sum of contributions of every call with a different size of list `lst` : $S = \sum_{i=0}^{L-1} \log_2(i) < L.\log_2(L)$. As there are $\alpha.L$ iterations instead of only L, this bound must be multiplied by α. So, the total cost of loop on `j` is $\mathcal{O}(\alpha.L.\log_2(L))$.

Finally, combining the costs of the L calls to `generateFrom()` and of the $\alpha.L$ iterations on `j`, we obtain a total cost of $\mathcal{O}\left(L.(N.4^N + \alpha.\log_2(L))\right)$ for the loop on `i`. This is clearly the dominant cost in the `while` loop, and thus the overall complexity of the transitive closure process. Also, the memory complexity is dominated by the list of generated cycles, leading to $\mathcal{O}(L.V)$.

Finally, it is worth noticing that for a given initial GC, the final set of GCs obtained by the transitive closure based on sub-sequence inversions may not be equal to the entire set of all possible GCs in the corresponding N-cube, even when there is no restriction over the size of the final set. This comes from the fact that some GCs may have no valid parallel transition for any transition in the sequence, as show in Figure 5. It is worth noticing that the example on the left of this figure is a binary-reflected Gray code (BRGC) [9, 7]. By construction, BRGCs do not contain any valid parallel transition, leading to no possible sub-sequence inversion. Moreover, this property is not restricted to BRGCs, as depicted in the example on the right of the figure. Indeed, such GCs with no valid parallel transition correspond to singletons in the partition of the entire set of GCs, based on the sub-sequence inversion relation. So, since the partition is composed of several connected components, the number of produced GCs will depend on the connected component of the initial GC. However, we show in Section 5 that this phenomenon is very limited in practice as the probability to get such a singleton as initial GC is very low and decreases with the dimension of the N-cube. Moreover, in such case, it is not a problem to generate another initial GC by using any classical generation algorithm. Thus, this fact is not a significant obstacle to the generation of as many codes as wanted.

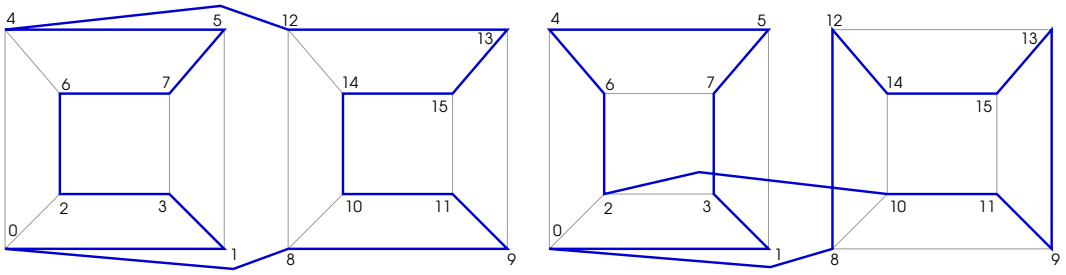


Figure 5: Cycles in the 4-cube with no valid parallel transition.

3 Using Gray Codes in Chaotic PRNGs

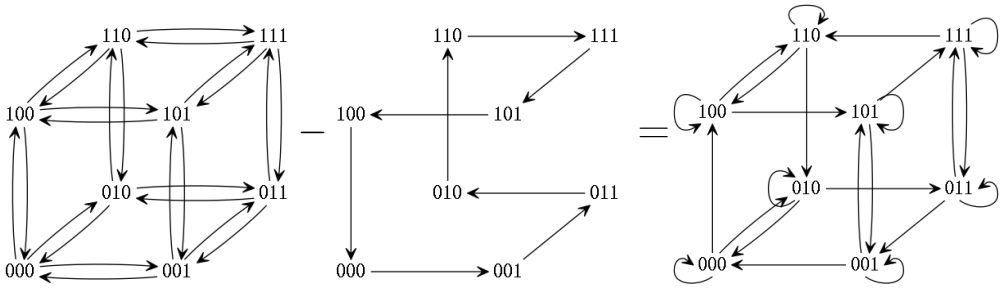
Full details on the use of Gray codes to build chaotic PRNGs can be found in [12, 11]. For the sake of clarity, we briefly recall the main principles in this section. The chaotic PRNG can be seen as an iterative post-processing of the output of a classical PRNG of N bits (possibly with weak statistical properties) as shown in Algorithm 3. The required elements are the initial binary vector $x^0 = (x_1^0, x_2^0, ..., x_N^0)$, a number b of iterations and an iteration function $f(x) = (f_1(x), ..., f_N(x))$. This function is used in the *chaotic mode* $F_f(x, s)$, where s is the index of the only bit of x that is updated by F, so that we have: $x^{t+1} = F_f(x^t, s_t) = (x_1^t, x_2^t, ..., f_s(x^t), ..., x_N^t)$.

The Gray code is used in the construction of function f: for a given dimension N, we start with the complete N-cube (Figure 6a), and we remove all the directed edges that belong to a given Gray code (Figure 6b) to obtain function f (Figure 6c). The removed edges are replaced by loop edges on every vertices.

Algorithm 3: Chaotic PRNG scheme.

```
Input: a function f, an iteration number b, an initial configuration x^0 (N bits)
Output: a configuration x (N bits)

X ← x^0
for i=0 to b−1 do
  s ← Random(N)
  X ← F_f(x, s)
endFor
return x
```



(a) Original 3-cube (b) A Gray code (c) Function f

Figure 6: Construction of function f in dimension 3.

Applying the function F_f on the pair (x, s) is equivalent to traverse the edge s from the node x in the modified N-cube presented in Figure 6c.

4 Reducing the Mixing Time Upper Bound of Boolean Functions in the N-Cube

As mentioned in the introduction and according to the PRNG presented in the previous section, the mixing time of a Boolean function in the N-cube provides important information about the minimal setting of the number of internal iterations of our chaotic PRNG (value b in Algorithm 3). In this section, a more accurate upper bound of this mixing time is proved by using Markov chains.

4.1 Theoretical Context

It has been proven in [12] that walking in an N-cube where an Hamiltonian cycle has been removed induces a Markov M chain whose stationary distribution π is the uniform one. Let us first formalize the relation between mixing time and uniform distribution.

First of all, let be given two distributions μ and π on the same set Ω, the Total Variation distance $\|\mu - \pi\|_{\text{TV}}$ is defined for by:

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

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

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

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 to the stationary distribution. It has been proven in ([18] Equation (4.36)) that $t_{\text{mix}}(\varepsilon) \leq \lceil \log_2(\varepsilon^{-1}) \rceil t_{\text{mix}}(\frac{1}{4})$. We are then left to evaluate $t_{\text{mix}}(\frac{1}{4})$.

For technical issues, rather than studying the random walk induced by the PRNG, we investigate the associated lazzy Markov chain, that is at each step, with probability 1/2 the chain moves to the next element and with probability 1/2 nothing is done. The mixing time of a lazzy Markov chain is about twice the mixing time of the initial chain ([5] p. 12).

Let N be a positive integer and $\Omega = \{0, 1\}^N$. For $X, Y \in \Omega$, we define the subset $H(X, Y)$ of $\{1, \ldots, N\}$ of indexes where X and Y differ: $H(X, Y) = \{i \mid X(i) \neq Y(i)\}$. We define the distance d on Ω by $d(X, Y) = |H(X, Y)|$, that is the number of components where X and Y differ.

We also consider a Hamiltonian path on the hypercube Ω. More precisely, let h be a bijective function from Ω to Ω inducing a cyclic permutation on Ω and such that for any X, $d(X, h(X)) = 1$.

Since $d(X, h(X)) = 1$, we denote by $f(X)$ the single index such that $X \neq h(X)$.

We consider the Markov Chain of matrix P on Ω defined by:

- $P(X, X) = \frac{1}{2} + \frac{1}{2N}$,
- $P(X, Y) = \frac{1}{2N}$ if $d(X, Y) = 1$ and $Y \neq h(X)$,
- $P(X, Y) = 0$ otherwise.

The Markov Chain P can be defined by the following random mapping representation: $X_{n+1} = r(X_n, U, V)$ where U is a uniform random variable on $1, \ldots, N$, V a uniform random variable on $\{0, 1\}$ and $r(X, U, V) = X$ if $U = f(X)$ and, otherwise, $r(X, U, V)$ is obtained from X by fixing its U-th index to V.

4.2 A Useful Random Process

The mixing time of the lazzy Markov chain will be studied in Section 4.3 using a classical coupling approach. In this section we propose a purely technical result useful for proving some properties on this coupling.

We consider a Markov chain (Z_i) on $\{0,\dots,N\}$, of matrix K defined by:

- $\mathbb{P}(Z_{i+1}=0 \mid Z_i=0)=1$,
- $\mathbb{P}(Z_{i+1}=0 \mid Z_i=1)=\frac{1}{2N}$,
- $\mathbb{P}(Z_{i+1}=2 \mid Z_i=1)=\frac{1}{N}$,
- $\mathbb{P}(Z_{i+1}=1 \mid Z_i=1)=1-\frac{3}{2N}$,
- $\mathbb{P}(Z_{i+1}=N \mid Z_i=N)=\frac{1}{N}$ and $\mathbb{P}(Z_{i+1}=N-1 \mid Z_i=N)=1-\frac{1}{N}$,
- for every $k\in\{2,\dots,N-1\}$, $\mathbb{P}(Z_{i+1}=k-1 \mid Z_i=k)=\frac{k-1}{N}$ and $\mathbb{P}(Z_{i+1}=k+1 \mid Z_i=k)=\frac{1}{N}$ and $\mathbb{P}(Z_{i+1}=k \mid Z_i=k)=1-\frac{k}{N}$,

Let $R_d=\min\{k \mid Z_k=0 \text{ and } Z_0=d\}$ if it exists and $\rho_d=E[R_d]$.

Proposition 1. *One has* $\rho_N\le 3N(1+e)+N\log(N-2)$.

Proof. According to Theorem 1.3.5 in [22], the ρ_i are the solution of the following system:

$$\begin{cases} \rho_N &= 1+\frac{1}{N}\rho_N+\frac{N-1}{N}\rho_{N-1}\\ \rho_i &= 1+\frac{1}{N}\rho_{i+1}+\frac{i-1}{N}\rho_{i-1}+(1-\frac{i}{N})\rho_i \quad \text{for } 2\le i\le N-1\\ \rho_1 &= 1+\frac{1}{2N}\rho_0+\frac{1}{N}\rho_2+(1-\frac{3}{2N})\rho_1\\ \rho_0 &= 0 \end{cases}$$

This system is equivalent to:

$$\begin{cases} \rho_N &= \frac{N}{N-1}+\rho_{N-1}\\ \frac{i}{N}\rho_i &= 1+\frac{1}{N}\rho_{i+1}+\frac{i-1}{N}\rho_{i-1} \quad \text{for } 2\le i\le N-1\\ \frac{3}{2N}\rho_1 &= 1+\frac{1}{N}\rho_2\\ \rho_0 &= 0 \end{cases}$$

Set, for $0\le i\le N-1$, $\alpha_i=\rho_{i+1}-\rho_i$.

Using the system, one has $\alpha_{N-1}=\frac{N}{N-1}$, $\alpha_0=\rho_1$. Moreover, for $2\le i\le N-1$, substituting ρ_{i+1} by $\alpha_i+\rho_i$ in the equation $\frac{i}{N}\rho_i=1+\frac{1}{N}\rho_{i+1}+\frac{i-1}{N}\rho_{i-1}$, one obtains that

$$\frac{i-1}{N}\rho_i=1+\frac{1}{N}\alpha_i+\frac{i-1}{N}\rho_{i-1}.$$

Therefore, for $2\le i\le N-1$,

$$\frac{i-1}{N}\alpha_{i-1}=1+\frac{1}{N}\alpha_i,$$

or, equivalently,

$$\alpha_{i-1}=\frac{N}{i-1}+\frac{1}{i-1}\alpha_i. \tag{1}$$

Since $\alpha_{N-1}=\frac{N}{N-1}$ and using Equation (1), one has

$$\alpha_{N-2}=\frac{N}{N-2}+\frac{N}{(N-1)(N-2)}$$

and

$$\alpha_{N-3}=\frac{N}{N-3}+\frac{N}{(N-2)(N-3)}+\frac{N}{(N-1)(N-2)(N-3)}.$$

By a direct induction, one obtains, for $1\le k\le N-1$:

$$\alpha_{N-k}=N\sum_{j=1}^{k}\prod_{\ell=0}^{j-1}\frac{1}{N-k+\ell}. \tag{2}$$

Consequently

$$\alpha_{N-k}\le N\sum_{j=1}^{k}\prod_{\ell=0}^{j-1}\frac{1}{N-k}=N\sum_{j=1}^{k}\frac{1}{(N-k)^j}.$$

It follows that, for $1\le k\le N-2$,

$$\alpha_{N-k}\le N\frac{1}{N-k}\frac{1-\left(\frac{1}{N-k}\right)^k}{1-\frac{1}{N-k}}=N\frac{1}{N-k-1}\left(1-\left(\frac{1}{N-k}\right)^k\right)\le\frac{N}{N-k-1}. \tag{3}$$

Now, for $k = N-1$, Equation (2) provides:

$$\alpha_1 = N \sum_{j=1}^{N-1} \frac{1}{j!}.$$

Therefore $\alpha_1 \leq N \sum_{j=0}^{N-1} \frac{1}{j!} \leq Ne$, where e is the Euler number.
Consequently and using Equation (3), we have

$$\sum_{k=1}^{N-1} \alpha_k \leq Ne + N \sum_{k=1}^{N-2} \frac{1}{N-k-1} = Ne + \sum_{k=1}^{N-2} \frac{1}{k}$$

Since the $N-2$th harmonic number $\sum_{k=1}^{N-2} \frac{1}{k}$ is lower than $1 + \log(N-2)$, we have

$$\sum_{k=1}^{N-1} \alpha_k \leq Ne + N\left(1 + \log(N-2)\right).$$

Moreover, by definition of the α_i's, $\sum_{k=1}^{N-1} \alpha_k = \rho_N - \rho_1$.
It follows that $\rho_N \leq \rho_1 + Ne + N\left(1 + \log(N-2)\right)$.
Since $\rho_2 - \rho_1 = \alpha_1 \leq Ne$ and $\frac{3}{2}\rho_1 = N + \rho_2$, one has $\frac{1}{2}\rho_1 \leq Ne + N$, proving that $\rho_1 \leq 2N(1+e)$. In conclusion,

$$\rho_N \leq \rho_1 + Ne + N\left(1 + \log(N-2)\right) \leq 3N(1+e) + N\log(N-2),$$

which concludes the proof. □

4.3 A Coupling for the Markov Chain P

We now define a coupling for the lazzy Markov chain. An upper bound of the mixing time is obtained by computing the maximal expected collapsing time of the coupling ([18] Chap. 5).

Let (U_n) be a sequence of i.i.d of random variables on $\{1, \ldots, N\}$, and (V_n) be a sequence of i.i.d of random variables on $\{0, 1\}$.

Let $(X_n, Y_n)_{n \geq 0}$ be a random sequence in $\Omega \times \Omega$ defined by:

$X_{n+1} = r(X_n, U_n, V_n)$, and $Y_{n+1} = X_{n+1}$ if $X_n = Y_n$ and, otherwise, $Y_{n+1} = r(Y_n, W_n, Z_n)$ where:

(1) If $f(X) \in H(X,Y)$ and $f(Y) \in H(X,Y)$, then $W_n = U_n$. Moreover, if $U_n \notin H(X,Y)$ then, $Z_n = V_n$, otherwise $Z_n = 1 - V_n$.

(2) If $f(X) \in H(X,Y)$ and $f(Y) \notin H(X,Y)$, then $W_n = U_n$. Moreover, if $U_n \notin H(X,Y)$ then, $Z_n = V_n$, otherwise $Z_n = 1 - V_n$.

(3) If $f(X) \notin H(X,Y)$ and $f(Y) \in H(X,Y)$, then $W_n = U_n$. Moreover, if $U_n \notin H(X,Y)$ then, $Z_n = V_n$, otherwise $Z_n = 1 - V_n$.

(4) If $f(X) \notin H(X,Y)$ and $f(Y) \notin H(X,Y)$, and $f(X) \neq f(Y)$. If $U_n \notin \{f(X), f(Y)\}$, then $W_n = U_n$. If $U_n = f(X)$, then $W_n = f(Y)$ and if $U_n = f(Y)$, then $W_n = f(X)$. Moreover, if $U_n \notin H(X,Y)$ then, $Z_n = V_n$, otherwise $Z_n = 1 - V_n$.

(5) If $f(X) \notin H(X,Y)$ and $f(Y) \notin H(X,Y)$, and $f(X) = f(Y)$, then $W_n = U_n$. Moreover, if $U_n \notin H(X,Y)$ then, $Z_n = V_n$, otherwise $Z_n = 1 - V_n$.

Proposition 2. *The sequence $(X_n, Y_n)_{n \geq 0}$ is a coupling for the Markov Chain P.*

Proof. By construction, (X_n) is a Markov Chain with transition matrix P. For (Y_n), it is clear that W_n follows the uniform distribution on $\{1, \ldots, N\}$. □

Lemma 1. *Let (d_n) be the sequence of integers defined by $d_n = d(X_n, Y_n)$. For any n, one has $|d_{n+1} - d_n| \leq 1$. Moreover:*

- *If $d_n \geq 2$, then $\mathbb{P}(d_{n+1} < d_n) \geq \frac{d_n - 1}{N}$.*
- *If $d_n \geq 2$, then $\mathbb{P}(d_{n+1} > d_n) \leq \frac{1}{N}$.*
- *If $d_n = 1$, then $\mathbb{P}(d_{n+1} = 0) \geq \frac{1}{2N}$.*
- *If $d_n = 1$, then $\mathbb{P}(d_{n+1} = 2) \leq \frac{1}{N}$ and $\mathbb{P}(d_{n+1} = 2) \leq \mathbb{P}(d_{n+1} = 0)$.*

Proof. Let $X = X_n$ and $Y = Y_n$. Assume first that $d_n \geq 2$, several cases may arise:

- If $f(X) \in H(X,Y)$ and $f(Y) \in H(X,Y)$ and $f(X) \neq f(Y)$. If $U_n \notin H(X,Y)$, then $d_{n+1} = d_n$. If $U_n \in H(X,Y)$ and $U_n \neq f(X)$ and $U_n \neq f(Y)$, then $d_{n+1} = d_n - 1$. If $U_n = f(X)$ or $U_n = f(Y)$, then $d_{n+1} = d_n$ with probability $\frac{1}{2}$ and $d_{n+1} = d_n - 1$ with probability $\frac{1}{2}$. It follows that, in this case, the probability that $d_{n+1} = d_n + 1$ is null. The probability that $d_{n+1} = d_n - 1$ is $\frac{|H(X,Y) \setminus \{f(X), f(Y)\}|}{N} + \frac{|\{f(X), f(Y)\}|}{2N} = \frac{d_n - 1}{N}$.

- If $f(X) \in H(X,Y)$ and $f(Y) \in H(X,Y)$ and $f(X) = f(Y)$. If $U_n \notin H(X,Y)$, then $d_{n+1} = d_n$. If $U_n \in H(X,Y)$ and $U_n \neq f(X)$, then $d_{n+1} = d_n - 1$. If $U_n = f(X)$, then $d_{n+1} = d_n$. The probability that $d_{n+1} = d_n - 1$ is $\frac{|H(X,Y)\setminus\{f(X)\}|}{N} = \frac{d_n - 1}{N}$.
- If $f(X) \in H(X,Y)$ and $f(Y) \notin H(X,Y)$. If $U_n \notin H(X,Y)$ and $U_n \neq f(Y)$, then $d_{n+1} = d_n$. If $U_n \in H(X,Y)$ and $U_n \neq f(X)$, then $d_{n+1} = d_n - 1$. If $U_n = f(X)$, then $d_{n+1} = d_n$ with probability $\frac{1}{2}$ and $d_{n+1} = d_n - 1$ with probability $\frac{1}{2}$. If $U_n = f(Y)$, then $d_{n+1} = d_n + 1$ with probability $\frac{1}{2}$ and $d_{n+1} = d_n$ with probability $\frac{1}{2}$. It follows that the probability that $d_{n+1} = d_n + 1$ is $\frac{1}{2N}$, and the probability that $d_{n+1} = d_n - 1$ is $\frac{|H(X,Y)\setminus\{f(X)\}|}{N} + \frac{|\{f(X)\}|}{2N} \geq \frac{d_n - 1}{N}$.
- If $f(X) \notin H(X,Y)$ and $f(Y) \in H(X,Y)$. This is a dual case of the previous one with similar calculus, switching $f(X)$ and $f(Y)$.
- If $f(X) \notin H(X,Y)$ and $f(Y) \notin H(X,Y)$, and $f(X) \neq f(Y)$. If $U_n \notin H(X,Y) \setminus \{f(X), f(Y)\}$, then $d_{n+1} = d_n$. If $U_n \in H(X,Y)$, then $d_{n+1} = d_n - 1$. If $U_n = f(X)$, then $d_{n+1} = d_n$ with probability $\frac{1}{2}$ and $d_{n+1} = d_n + 1$ with probability $\frac{1}{2}$. Similarly, If $U_n = f(Y)$, then $d_{n+1} = d_n$ with probability $\frac{1}{2}$ and $d_{n+1} = d_n + 1$ with probability $\frac{1}{2}$. Consequently, the probability that $d_{n+1} = d_n + 1$ is $\frac{1}{2N} + \frac{1}{2N} = \frac{1}{N}$. Moreover the probability that $d_{n+1} = d_n - 1$ is $\frac{|H(X,Y)|}{N} = \frac{d_n}{N} \geq \frac{d_n - 1}{N}$.
- If $f(X) \notin H(X,Y)$ and $f(Y) \notin H(X,Y)$, and $f(X) = f(Y)$. If $U_n \in H(X,Y)$, then $d_{n+1} = d_n - 1$. If $U_n \notin H(X,Y)$, then $d_{n+1} = d_n$. It follows that the probability that $d_{n+1} = d_n + 1$ is null and the probability that $d_{n+1} = d_n - 1$ id $\frac{|H(X,Y)|}{N} = \frac{d_n}{N} \geq \frac{d_n - 1}{N}$.

Now assume that $d_n = 1$. By definition, $|H(X,Y)| = 1$. The following cases arise:

- If $f(X) = f(Y)$. Since f is induced by an Hamiltonian cycle and $N \geq 2$, we cannot have $H(X,Y) = \{f(X)\}$. If $U_n \notin H(X,Y)$, then $d_{n+1} = d_n$. If $U_n \in H(X,Y)$, $d_{n+1} = d_n - 1$. Therefore, the probability that $d_{n+1} = d_n + 1$ is null and the probability that $d_{n+1} = d_n - 1 = 0$ is $\frac{1}{N}$.
- If $f(X) \neq f(Y)$ and $H(X,Y) = \{f(X)\}$. If $U_n \neq f(X)$ and $U_n \neq f(Y)$, then $d_{n+1} = d_n$. If $U_n = f(X)$, then $d_{n+1} = 0$ with probability $\frac{1}{2}$ and $d_{n+1} = d_n = 1$ with probability $\frac{1}{2}$. If $U_n = f(Y)$, then $d_{n+1} = d_n + 1 = 2$ with probability $\frac{1}{2}$ and $d_{n+1} = d_n = 1$ with probability $\frac{1}{2}$. Therefore, the probability that $d_{n+1} = 2$ is $\frac{1}{2N}$ and the probability that $d_{n+1} = 0 = \frac{1}{2N}$.
- If $f(X) \neq f(Y)$ and $H(X,Y) = \{f(Y)\}$. It a dual case of the previous one. The probability that $d_{n+1} = 2$ is $\frac{1}{2N}$ and the probability that $d_{n+1} = 0 = \frac{1}{2N}$.
- If $f(X) \neq f(Y)$ and $H(X,Y) \cap \{f(X), f(Y)\} = \emptyset$. If $U_n \neq f(X)$ and $U_n \neq f(Y)$, then $d_{n+1} = d_n$. If $U_n = f(X)$ (resp. $f(Y)$), then $d_{n+1} = d_n + 1 = 2$ with probability $\frac{1}{2}$ and $d_{n+1} = d_n = 1$ with probability $\frac{1}{2}$. Therefore, the probability that $d_{n+1} = 2$ is $\frac{1}{N}$ and the probability that $d_{n+1} = 0 = \frac{1}{N}$.

□

Let $T(X,Y)$ the random variable of the first coupling time of (X_n, Y_n), with $X_0 = X$ and $Y_0 = Y$, i.e., $T(X,Y) = k$ if $X_k = Y_k$ and $X_i \neq Y_i$ for $i < k$. Let $\tau(X,Y) = E[T(X,Y)]$ and $\tau = max_{X,Y}\tau(X,Y)$.

Proposition 3. *One has $\tau \leq \rho_N$.*

Proof. By Proposition 1 and Lemma 1. □

By classical results on coupling ([18] Corollary 5.3) and Markov inequality, one has $d(t) \leq \frac{\tau}{t}$. Thus, $t_{\text{mix}}(1/4) \leq 4\tau$ and finally

$$t_{\text{mix}}(\varepsilon) \leq \lceil \log_2(\varepsilon^{-1}) \rceil \left(12N(1+e) + 4N\log(N-2)\right). \tag{4}$$

5 Experiments

In this section is presented a set of different experiments in relation to the GC generation algorithm and the mixing time.

5.1 Performance of Gray Codes Generation Algorithms

The first series of experiments is devoted to the performance of the GC generation algorithm. The execution times shown below have been obtained on a desktop machine with two Intel(R) Xeon(R) CPU E5-2640 v2 @ 2.00GHz containing eight cores each and 128GB RAM. The OS is Linux Debian 3.16.43 and the program is written in C/C++ and uses the OpenMP multi-threading library. Indeed, the implementation of our algorithm is multi-threaded and takes advantage of the available cores in the machine. This reduces the execution times but does not change the overall complexity of the algorithm. Also, different compilers have been tested (GCC, Intel, PGI) and although slight variations

may be observed, the executions times are quite similar. Times given below are averages of a series of executions.

Two series of results are presented that exhibits the behavior of the performance our algorithm either in function of the dimension of the N-cube or in function of the number of GCs to generate.

In Table 1 are provided the execution times for generating 100,000 cycles in N-cubes of dimensions 5 to 10. Smaller dimensions have too few GCs to get representative times (only one GC in dimensions 2 and 3 and 9 in dimension 4). As expected, the execution time increases with the dimension. However, the progression is much slower than the theoretical complexity obtained in Section 2.4. This means that this upper bound of complexity is largely overestimated. Indeed, many iterations in the inner loop of the generation algorithm are not performed, since it stops as soon as a parallel transition has been found. This is not taken into account in the current estimation of the complexity but it will deserve a deeper study to get a more accurate expression.

dim	5	6	7	8	9	10
time (s)	19.61	19.93	28.17	53.60	174.43	332.29

Table 1: Generation times of 100,000 Gray Codes (GCs) in function of the dimension of the N-cube.

The execution times for generating different amounts of GCs in dimensions between 5 and 9 are given in Figure 7. We observe similar behaviors for all dimensions with a sharp increase of the execution time with the number of GCs. This mainly comes from the fact that in the closure algorithm, GCs may be generated as much times as their number of neighbors according to the sub-sequence inversion relation. Moreover, this number of duplicate generations increases with the size of the input list, which itself increases with the size of the generated set. This is why we observe such an exponential increase. A strategy to avoid such duplicates during the transitive closure will be investigated in future works. Nonetheless, the absolute performance of the current version is rather efficient when considering that the production of several thousands of GCs takes less than one minute.

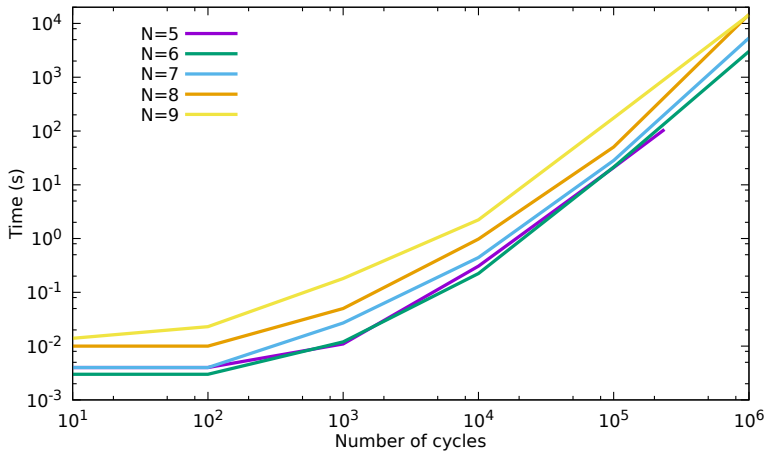


Figure 7: Exec times (s) in function of the number of generated GCs for different dimensions of N-cube.

To conclude this part on the performance of the generation process, a first multi-threaded implementation has been tested, which provides gains between 16% and 90% depending on the number of generated cycles and the dimension of the N-cube. However, this first implementation is far from being optimal and additional development effort is planed as a future work.

5.2 Statistical Study of the Generation Algorithm

This second series of experiments aims at analyzing the impact of the presence of isolated GCs according to the sub-sequence inversion relation, as mentioned in Section 2.4. Table 2 gives the percentages of isolated GCs in dimensions 2 to 5 of N-cubes. It is not possible to get exact percentages for greater dimensions due to the exponential increase of the number of GCs. However, it is interesting to see that the ratio of isolated GCs drastically decreases when the dimension of the N-cube increases. It can be proved that this tendency continues with greater dimensions. This comes from the condition that a GC must satisfy to be isolated, i.e., to generate no other GC by the inversion algorithm. The condition is that it must not have any valid parallel transitions in its entire sequence.

However, for one transition, the number of possible parallel transitions is directly linked to the dimension of the N-cube. Moreover, the number of transitions in a sequence exponentially increases with the dimension of the N-cube. Therefore, the probability of having at least one valid parallel transition in the entire sequence exponentially increases with the dimension of the N-cube. Indeed, the constraint to satisfy becomes stronger and stronger as the dimension increases. Thus, it comes that the probability that a GC is isolated, i.e., is a singleton in the partition of the set of GCs, exponentially decreases when the dimension of the N-cube increases. Finally, this implies that the constraint over the choice of GCs as input to the closure algorithm is very weak, as most GCs will lead to the generation of other GCs.

dim	2	3	4	5
nb of isolated GCs	1	1	3	740
total nb of GCs	1	1	9	237,675
ratio of isolated (%)	100	100	33.3	0.3

Table 2: Percentages of isolated GCs in the partition of the set of GCs.

5.3 Discussion over the Mixing Time Distributions in Dimensions 5 and 8

First of all, the mixing time is bounded by $\lceil \log(\varepsilon^{-1}) \rceil \rho_N$ and thus, thanks to Proposition 1 by $\lceil \log(\varepsilon^{-1}) \rceil 3N(1+e) + N\log(N-2)$.

For instance, for $\varepsilon = 10^{-6}$, $N = 5$, or $N = 8$ these bounds are respectively equal to 319 and 550. In the same context, i.e., $\varepsilon = 10^{-6}$, we practically observe that the largest mixing time for all the Markov chains when $N = 5$ is equal to 42. It can be thus notably deduced that theoretical bound provided in Proposition 1 is notably much larger than the practical one.

6 Conclusions

Practical and theoretical contributions have been presented in this paper, respectively related to the field of Gray codes generation and mixing time of Boolean functions in dimension N.

Concerning Gray codes generation, an algorithm based on sub-sequence inversion has been designed and implemented. It starts from one known GC and it produces new GCs of the same dimension. It has been explained why the number of generations strongly depends on the configuration of the input GC. Then, as the upper bound of the number of generations from a single GC is limited, a transitive closure algorithm has been proposed too, that can generate very large subsets of the entire set of GCs in a given N-cube. Such algorithms are essential to study the Gray Code properties that may present a particular interest for use in a chaotic PRNG based on random walks or jumps in a N-cube.

Experiments have shown that our algorithm can produce several thousands of GCs in time of the order of the minute. Also, it has been shown that for a given dimension of N-cube, the number of isolated GCs, i.e., the codes that do not allow any sub-sequence inversion, is a negligible part of the total number of GCs. Moreover, this ratio decreases when the dimension increases. Therefore, the probability that our algorithm does not produce any new GC from a given input, is very small.

Concerning the mixing time of Boolean functions in the N-cube, the complete proof of an upper bound in $\mathcal{O}\left(N.\log_2(N)\right)$ has been given. This is the most accurate upper bound that has been proved to this day.

Such result is useful to limit significantly the number of iterations when computing mixing times of Boolean functions in the N-cube. This is of particular interest when looking for such functions that would be adequate for the use in a CI-PRNG, i.e., functions with minimal mixing time, among a large number of candidates.

In future works, we plan to address the potential algorithmic enhancements mentioned in the experiments, and to perform deep statistical analyzes of Gray codes according to the mixing times of their respective Boolean functions in the N-cube as well as other features. This will help to identify the particular properties of GCs that could be used to find the codes best suited to the use in PRNGs. The results presented in this paper directly contribute to make such statistical analyzes possible.

Acknowledgments: This work is partially funded by the Labex ACTION program (contract ANR-11-LABX-01-01).

References

[1] Jacques M. 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 Jan Arne Telle, editors, *Fundamentals of Computation Theory*, pages 126–137, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[2] Mohammed Bakiri. *Hardware Implementation of Pseudo Random Number Generator Based on Chaotic Iterations*. Theses, Université Bourgogne Franch-Comté, January 2018.

[3] Mohammed Bakiri, Jean-François Couchot, and Christophe Guyeux. FPGA implementation of f2-linear pseudorandom number generators based on zynq mpsoc: A chaotic iterations post processing case study. In Christian Callegari, Marten van Sinderen, Panagiotis G. Sarigiannidis, Pierangela Samarati, Enrique Cabello, Pascal Lorenz, and Mohammad S. Obaidat, editors, *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT, Lisbon, Portugal, July 26-28, 2016.*, pages 302–309. SciTePress, 2016.

[4] Mohammed Bakiri, Jean-François Couchot, and Christophe Guyeux. CIPRNG: A VLSI family of chaotic iterations post-processings for $\mathbb {F}_{2}$ -linear pseudorandom number generation based on zynq mpsoc. *IEEE Trans. on Circuits and Systems*, 65-I(5):1628–1641, 2018.

[5] Nathanaell Berestycki. Mixing times of markov chains: Techniques and examples. Graduate course at cambridge, University of Cambridge, 2016.

[6] Girish S. Bhat and Carla D. Savage. Balanced gray codes. *Electr. J. Comb.*, 3(1), 1996.

[7] Martin W. Bunder, Keith P. Tognetti, and Glen E. Wheeler. On binary reflected gray codes and functions. *Discrete Mathematics*, 308(9):1690–1700, 2008.

[8] I. S. Bykov. On locally balanced gray codes. *Journal of Applied and Industrial Mathematics*, 10(1):78–85, 2016.

[9] Marston D. E. Conder. Explicit definition of the binary reflected gray codes. *Discrete Mathematics*, 195:245–249, 1999.

[10] Sylvain Contassot-Vivier and Jean-François Couchot. Canonical Form of Gray Codes in N-cubes. In Alberto Dennunzio, Enrico Formenti, Luca Manzoni, and Antonio E. Porreca, editors, *23th International Workshop on Cellular Automata and Discrete Complex Systems (AUTOMATA)*, volume LNCS-10248 of *Cellular Automata and Discrete Complex Systems*, pages 68–80, Milan, Italy, Jun 2017. Springer International Publishing. Part 2: Regular Papers.

[11] Sylvain Contassot-Vivier, Jean-François Couchot, Christophe Guyeux, and Pierre-Cyrille Héam. Random Walk in a N-Cube Without Hamiltonian Cycle to Chaotic Pseudorandom Number Generation: Theoretical and Practical Considerations. *International Journal of Bifurcation and Chaos*, 27(01):18 pages, Jan 2017.

[12] Jean-François Couchot, Pierre-Cyrille Héam, Christophe Guyeux, Qianxue Wang, and Jacques M. Bahi. Pseudorandom number generators with balanced gray codes. In *SECRYPT 2014 - Proceedings of the 11th International Conference on Security and Cryptography, Vienna, Austria, 28-30 August, 2014*, pages 469–475, 2014.

[13] Persi Diaconis, Ronald L. Graham, and John A. Morrison. Asymptotic analysis of a random walk on a hypercube with many dimensions. *Random Structures & Algorithms*, 1(1):51–72, 1990.

[14] Christophe Guyeux, Raphaël Couturier, Pierre-Cyrille Héam, and Jacques M. Bahi. Efficient and cryptographically secure generation of chaotic pseudorandom numbers on GPU. *The Journal of Supercomputing*, 71(10):3877–3903, 2015.

[15] Gregory F Lawler and Vlada Limic. *Random walk: a modern introduction*, volume 123. Cambridge University Press, 2010.

[16] Pierre L'Ecuyer. History of uniform random number generation. In *WSC*, pages 202–230. IEEE, 2017.

[17] Pierre L'Ecuyer and Richard Simard. Testu01: A c library for empirical testing of random number generators. *ACM Transactions on Mathematical Software (TOMS)*, 33(4):22:1–22:40, Aug 2007.

[18] David A Levin and Yuval Peres. *Markov chains and mixing times*, volume 107. American Mathematical Soc., 2017.

[19] László Lovász. Random walks on graphs. *Combinatorics, Paul erdos is eighty*, 2:1–46, 1993.

[20] P. L'Ecuyer. *Handbook of Computational Statistics*, chapter Random Number Generation, pages 35–71. Springer-Verlag, 2nd edition edition, 2012.

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

[22] J. R. Norris. *Markov Chains*. Cambridge University Press, 1998. ISBN: 9780521633963.

[23] John P. Robinson and Martin Cohn. Counting sequences. *IEEE Trans. Comput.*, 30(1):17–23, jan 1981.

[24] Benedetto Scoppola. Exact solution for a class of random walk on the hypercube. *Journal of Statistical Physics*, 143(3):413–419, 2011.

[25] IN Suparta and AJ van Zanten. Totally balanced and exponentially balanced gray codes. *Discrete Analysis and Operation Research (Russia)*, 11(4):81–98, 2004.

[26] IN Suparta and AJ van Zanten. A construction of gray codes inducing complete graphs. *Discrete Mathematics*, 308(18):4124–4132, 2008.

[27] Marcel Wild. Generating all cycles, chordless cycles, and hamiltonian cycles with the principle of exclusion. *Journal of Discrete Algorithms*, 6(1):93 – 102, 2008. Selected papers from {AWOCA} 2005Sixteenth Australasian Workshop on Combinatorial Algorithms.