Naive Czyzewski Generator, a different approach[1]

Maciej A. Czyzewski

June 6, 2015

[1] This is the official paper, there is also an article with the same title which loosely presents a generator.

`http://maciejczyzewski.me/2015/06/05/naive-czyzewski-generator.html`

This document describes the NCG randomization function. It also provides examples, comments and analysis.

The generator was developed to work as PSEUDO-RANDOM NUMBER GENERATOR, HASH FUNCTION or a CRYPTOGRAPHIC MODULE. This document aims to be a demonstration of the features of the algorithm and a guide to their use.

Introduction

It is introduced a new randomization function named NAIVE CZYZEWSKI GENERATOR. An implemented algorithm has a period between $2^{n/2}$ to 2^n, excellent quality, and seems to be best among all generators ever invented.[2]

[2] Source code is available in Implementation section (page 6).

Structure

The algorithm consists of three basic functions:

- Push — Throws 32-bit value, otherwise known as seed, to CURRENT STATE MATRIX.[3] During this operation, script employs all matrix cells, by changing their values, and creates a new I value.[4]
- Pull — It returns 32-bit block of output, generated by algorithm. While creating this value, 4 states from matrix are modified. Followed by the PAIR JUMP, which modifies the value $I = I + 2$.
- Reset — This function replaces current matrix by the INITIAL STATE MATRIX. There is no need to modify I value.

[3] In the later part of this document you will find out what it is.

[4] This is called as TRANSIENT INCREMENT.

The generator structure forms some kind of universal stochastic model, based on CURRENT STATE MATRIX, for better understanding let colorize cells in two colors - *black and white*. (figure 1)

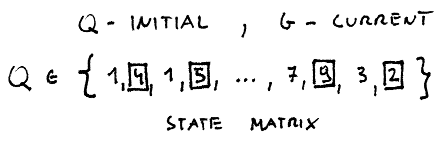
Q - INITIAL , G - CURRENT
$Q \in \{1, \boxed{4}, 1, \boxed{5}, \ldots, 7, \boxed{9}, 3, \boxed{2}\}$
STATE MATRIX

Figure 1: State matrix. *Where black cells are in squares.*

In default version INITIAL STATE MATRIX has 16 cells, and it is filled with consecutive digits of π after the decimal point.

During initialization, INITIAL STATE MATRIX is copied to CURRENT STATE MATRIX. All operations in the algorithm are done on the current matrix. Therefore, change in the initial values, changes the output sequence.

Algorithm

Before we move on to discuss the algorithm, let's define 3 basic macros of NCG written in C language. First is $M(i)$, which modulates an iterator to the size of the INITIAL STATE MATRIX, for example $G[M(i)] = G_i$. Second is known as rotation formula.[5] Last macro binds together two numbers and modifies their states.

[5] Joel Yliluoma. *Synthesizing arithmetic operations using bit-shifting tricks*. http://bisqwit.iki.fi/, 2014

```
// Abbreviation for getting values from the matrix
#define M(i) ((i) & (SIZE - 1))

// Bit rotation
#define R(x, y) (((x) << (y)) | ((x) >> (16 - (y))))

// XOR gate, relationships
#define L(x, y) {                                      \
  (y) ^= ((x) << 5) ^ ((y) >> 3) ^ ((x) >> 1); \
  (x) ^= ((y) << 8) ^ ((x) << 3) ^ ((y) << 9); \
}
```

Push function is designed to transform existing matrix to different matrix, using a seed value. It is based on a simple reinforcement of bits. Additionally, it creates new I value, which determines the color of the cells.[6]

[6] When I is an odd number, the first cell will be black.

```
Data: seed, G matrix
Result: new I, changed G matrix
/* Preparation                                                    */
$S = seed;$
$I = S * 0x3C6EF35F;$
for $i = 0$ to G length do
    /* Reinforcement                                              */
    $G_i = G_i \oplus ((S * I - I \oplus S) \gg 16);$
    $G_i = G_i \oplus ((S * I + I \oplus S) \gg 00);$
    /* Finalization                                               */
    $I = I \oplus (((G_{I-1} + G_i) \ll 16) \oplus ((G_{I+1} - G_i) \ll 00));$
end
```

Algorithm 1: push function

Pull function is most important for us, this function returns a random number. At the beginning we get 5 states (figure 2) from the generator: three white, one black, and one chaotic - with equal probability it may be white or black. Before returning the random value, we will do PAIR JUMP, thus ending the execution of the procedure and loops this mechanism. That is why we call I as TRANSIENT INCREMENT.

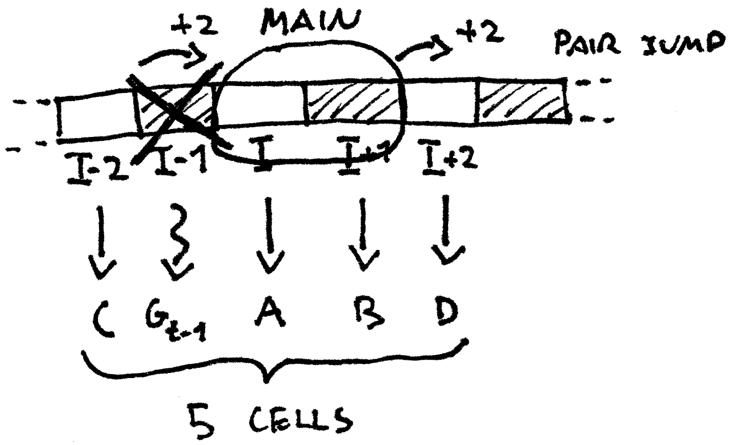

Figure 2: 5 cells. *With the marked names.*

Please note that we do not use G_{I-1} cell, instead we use chaotic G_{t-1}, whose selection depends on the value in the generator.

```
Data: $I$, $G$ matrix
Result: new $I$, changed $G$ matrix, random $t$
/* Variebles                                                    */
$a = G_I$; $b = G_{I+1}$;
$c = G_{I-2}$; $d = G_{I+2}$;
/* Initialization                                               */
$t = (a + I) * (b - S)$;
/* Allowance                                                    */
$t = t \oplus (a \oplus (b \ll 8) \oplus (c \ll 16) \oplus (d \& 0xff) \oplus ((d \gg 8) \ll 24))$;
/* Mixing                                                       */
$L(G_I, G_{I-2})$;
$L(G_I, G_{I+2})$;
/* Transformation                                               */
$G_I = G_{t-1} \oplus R(c, M(t)) \oplus R(d, M(t)) \oplus a$;
$G_{I+1} = (b \gg 1) \oplus (-(b \& 1u) \& 0xB400u)$;
/* Increment                                                    */
$I = I + 2$;
return $t$;
```

Algorithm 2: pull function

Black cells in this implementation are modified by LFSR[7], but it can be any other function or generator, as well. It is just to guarantee period length and uniqueness of cycle.

[7] Kewal K. Saluja. *Linear Feedback Shift Registers Theory and Applications*. University of Wisconsin-Madison, 1987

Period

Half of the cells has a known period 2^{16}, half unknown.[8] So we can approximately determine how much it is. (s - number of cells)

[8] LFSR have period 2^n, but in this case cells have 16-bit, which gives period 2^{16}.

$$\prod^{s/2} 2^{16} \leq x \leq \prod^{s} 2^{16}$$
$$2^{8s} \leq x \leq 2^{16s}$$

In the reference implementation we have $s = 16$, which gives the period 2^{128}, which can be up to 2^{256}.

Features

The advantages of this approach can be presented in points:

- Initial state matrix — May be modified, what does not change the qualities of algorithm, but changes the series of output.

- Arbitrary period — Period is known, between $2^{n/2}$ to 2^n, it depends only on the amount of cells in matrix and chosen seed.
- High-quality output — It has a very uniform distribution, therefore it is proof to many attacks, extremely easily passes the DIEHARD and TESTU01 tests.
- Cryptographically secure — There is no known successful attack.
- Flexible skeleton — Function that operates on black cells, can be changed. As was mentioned, now it is LFSR because it's period is determined.
- Suitable speed — Quite good comparing to the other benefits.
- Universal — Due to it's simple design, can be used for any purpose.

Analysis

Since PRNGS are commonly used for cryptographic purposes, it is sometimes required that the transformation and output functions satisfy two additional properties:

- Unpredictability — Given a sequence of output bits, the preceding and following bits shouldn't be predictable.
- Nonreversibility — Given the current state of the generator, the previous states shouldn't be computable.

NCG meets these conditions. Mapping the result to bitmap shows how evenly the data is distributed. Generator passes the DIEHARD and TESTU01 tests. It meets the conditions of security requirements for cryptographic modules.

Proposal

Numbers generated by NCG are useful in many different kinds of applications: [9]

[9] Donald E. Knuth. *The Art of Computer Programming*. Four volumes (planned seven). Addison-Wesley, 1968. ISBN 0-201-03801-3

- Cryptography — One of the most difficult aspect of cryptographic algorithms is in depending on, or generating, true random information. A source of unbiased bits is crucial for many types of secure communications, when data needs to be concealed.
- Simulation — When a computer is being used to simulate natural phenomena, economic, meteorology, nuclear physics and many other models, random numbers are required to make things realistic.

- Sampling — Sampling distributions are important in statistics because they provide a major simplification en route to statistical inference. It is often impractical to examine all possible cases, but a random sample will provide insight into what constitutes typical behavior.
- Computer programming — Random values make a good source of data for testing the effectiveness of computer algorithms. They are crucial to the operation of randomized algorithms, which are often far superior to their deterministic counterparts.
- Decision making — Randomness is also an essential part of optimal strategies in the theory of games. There are reports that many executives make their decisions by flipping a coin or by throwing darts, etc.
- Numerical analysis — Ingenious techniques for solving complicated numerical problems have been devised using random numbers. There is extensive use of random numbers in such areas as numerical analysis for Monte Carlo and Quasi-Monte Carlo integration.

Implementation

If you are interested in reference implementation, you can download it from `http://maciejczyzewski.me/assets/files/NCG.zip`. The whole code is written in ANSI C.[10]

[10] The implementation of the algorithm available in Appendix section (page 7).

Support

The website for the NCG is located at `http://maciejczyzewski.me/2015/06/05/naive-czyzewski-generator.html`. Here, you'll find links to GIT repository, this paper, and other my work.

References

Donald E. Knuth. *The Art of Computer Programming*. Four volumes (planned seven). Addison-Wesley, 1968. ISBN 0-201-03801-3.

Kewal K. Saluja. *Linear Feedback Shift Registers Theory and Applications*. University of Wisconsin-Madison, 1987.

Joel Yliluoma. *Synthesizing arithmetic operations using bit-shifting tricks*. http://bisqwit.iki.fi/, 2014.

Appendix

```
/*  NCG written in 2015 by Maciej A. Czyzewski

To the extent possible under law, the author has dedicated all copyright
and related and neighboring rights to this software to the public domain
worldwide. This software is distributed without any warranty.

See <http://creativecommons.org/publicdomain/zero/1.0/>.  */

#include <stdint.h>
#include <string.h>

// S - seed, I - increment, t - mask, i - temporary
uint32_t S, I, t, i;

// The length of the initial states
#define SIZE 16

// Abbreviation for getting values from the matrix
#define M(i) ((i) & (SIZE - 1))

// Bit rotation
#define R(x, y) (((x) << (y)) | ((x) >> (16 - (y))))

// XOR gate, relationships
#define L(x, y) {                                        \
  (y) ^= ((x) << 5) ^ ((y) >> 3) ^ ((x) >> 1);          \
  (x) ^= ((y) << 8) ^ ((x) << 3) ^ ((y) << 9);          \
}

// Variebles in the algorithm
uint16_t a, b, c, d;

// Initial state matrix (pi digits)
uint16_t G[SIZE], Q[SIZE] = { 1, 4, 1, 5, 9, 2, 6, 5,
                              3, 5, 8, 9, 7, 9, 3, 2 };

void push(uint32_t seed) {
  // Preparation
  I = seed * 0x3C6EF35F;

  for (S = seed, i = 0; i < SIZE; i++) {
    // Reinforcement
    G[M(i)] ^= (S * I - I ^ S) >> 16;
    G[M(i)] ^= (S * I + I ^ S) >> 00;

    // Finalization
    I ^= ((G[M(I - 1)] + G[M(i)]) << 16)
      ^  ((G[M(I + 1)] - G[M(i)]) << 00);
  }
}

uint32_t pull(void) {
  // Variebles
  a = G[M(I + 0)]; b = G[M(I + 1)];
  c = G[M(I - 2)]; d = G[M(I + 2)];

  // Initialization
  t = (a + I) * (b - S);

  // Allowance
  t ^= a ^ (b << 8) ^ (c << 16) ^ (d & 0xff) ^ ((d >> 8) << 24);

  // Mixing
  L(G[M(I + 0)], G[M(I - 2)]);
  L(G[M(I + 0)], G[M(I + 2)]);

  // Transformation
  G[M(I + 0)] = G[M(t - 1)] ^ R(c, M(t)) ^ R(d, M(t)) ^ a;
  G[M(I + 1)] = (b >> 1) ^ (-(b & 1u) & 0xB400u); // LFSR

  // Increment
  I += 2;

  return t;
}

void reset(void) {
  // Copying defaults
  memcpy(G, Q, 2 * SIZE);
}
```