Przegląd wybranych generatorów liczb pseudolosowych
i ich analiza pod kątem losowości.

MATEUSZ SOŁTYSIK, ANDRZEJ KWAK, WIKTOR DYNGOSZ
Politechnika Wrocławska, *Wydział Podstawowych Problemów Techniki*
May 30, 2014

Abstract

Keywords: PRGN, χ^2

Wstęp

Pseudo-Random Number Generator (PRNG) – program lub podprogram, który na podstawie niewielkiej ilości informacji (ziarno, ang. seed) generuje deterministyczny, potencjalnie nieskończony, ciąg liczb. Generatory liczb pseudolosowych nie generują ciągów prawdziwie losowych - generator inicjowany ziarnem, które może przyjąć k różnych wartości, jest w stanie wyprodukować co najwyżej k różnych ciągów liczb. Ponieważ rozmiar zmiennych reprezentujących wewnętrzny stan generatora jest ograniczony i może on znajdować się tylko w ograniczonej liczbie stanów, po pewnym czasie generator dokona pełnego cyklu i zacznie generować te same wartości. Teoretyczny limit długości cyklu wyrażony jest przez 2^n, gdzie n to liczba bitów przeznaczonych na przechowywanie stanu wewnętrznego. W praktyce, większość generatorów ma znacznie krótsze okresy.
Pożądane cechy generatorów:

1. trudne do ustalenia ziarno, choć znany jest ciąg wygenerowanych bitów
2. trudne do ustalenia kolejno generowane bity, choć znany jest ciąg bitów dotychczas wygenerowanych

1 Opis i analiza algorytmów PRNG

Informacje wstępne

Seed (pl. ziarno) - wartość liczbowa, którą inicjujemy generator. Z racji tego, iż PRNG są deterministyczne, wybrany algorytm dla danego seeda generuje identyczne wyjście.

```
public static long getSeed() {
      SecureRandom random = new SecureRandom();
      byte seed[] = random.generateSeed(20);
      return Longs.fromByteArray(seed);
}
```

Listing 1: Przykładowa funkcja generująca seeda

Testy chi-kwadrat

Testy spektralne

Testy dla każdego algorytmu będą przeprowadzane dla czterech wartości: 10^1, 10^3, 10^6, 10^9, bo zakres 32-bitowej liczby $= \frac{32\log(2)}{\log(10)} \approx 9$.

1.1 Blum Blum Shub

Algorytm *Blum Blum Shub* został zaproponowany przez Lenore Blum, Manuela Blum'a oraz Michaela Shub'a wiosną, 1986. roku, w pracy: "A Simple Unpredictable Pseudo-Random Number Generator"[1]. Generator ten jest postaci:

$$x_{n+1} = (x_n)^2 \bmod N$$

gdzie x_{n+1} to kolejny stan generatora, N to iloczyn dwóch dużych liczb pierwszych p i q takich, że:

1. dają w dzieleniu przez 4 resztę 3 ($p \equiv q \equiv 3 \bmod 4$),
2. mają możliwie mały $NWD(\phi(p-1), \phi(q-1))$, co zapewnia długi cykl.

Wynikiem generatora jest kilka ostatnich bitów x_n.

```
private static final int p = 11;
private static final int q = 19;

public static long getRandomNumber() {
    seed = (seed * seed) % (p * q);
    return Math.abs(seed);
```

[1] http://epubs.siam.org/doi/abs/10.1137/0215025

```
}
```

Listing 2: Generowanie następnej liczby pseudolosowej przez BBS

Generator ten jest dosyć wolny, ale za to bardzo bezpieczny. Przy odpowiednich założeniach, odróżnienie jego wyników od szumu jest równie trudne co faktoryzacja N.

1.2 Liniowy Generator Kongruentny (LCG)

```
private final static long a = 25173;
private final static long b = 13849;
private final static long m = 32768;

public static long getRandomNumber() {
    seed = (a * seed + b) % m;
    return seed;
}
```

Listing 3: Generowanie następnej liczby pseudolosowej przez LCG

Addytywny LCG $xi + 1 = (a * xi + c)modm$, Multiplikatywny LCG $xi + 1 = a * ximodm$ ••Generowane kolejno liczby są z zakresu od 0 do c-1, stąd po m cyklach obliczeniowych liczby pseudolosowe zaczynają się powtarzać Dla pewnych kombinacji parametrów generowany ciąg jest prawie losowy, dla innych bardzo szybko staje się okresowy •Okres obu przedstawionych generatorów zależy od wartości parametrów równania i opisują twierdzenia: generatora liniowego wynosi N = 2k-2, gdy a = 3 mod 8 lub a = 5 mod 8. – Jeżeli m jest liczbą pierwszą, to generator liniowy posiada okres maksymalny równy m, gdy a jest pierwiastkiem pierwotnym m – Pierwiastek pierwotny modulo n to liczba z przedziału

Algorytm doboru współczynników dla generatora LCG Określamy zakres liczb pseudolosowych 0..xmax dla LCG addytywnego 1..xmax dla LCG multiplikatywnego Moduł m jest zawsze o 1 większy od maksymalnej liczby w zakresie, czyli: m = x max + 1 Przyrost c musi być względnie pierwszy z modułem m – moduł m można rozłożyć na czynniki pierwsze i jako c wybrane mogą być czynniki nie występujące w rozłożeniu
– c może być generowane pseudolosowe, pod warunkiem, że spełnia warunek: gcd(c,m) = 1
Mnożnik a dobierany jest tak, aby a - 1 było podzielne przez każdy czynnik pierwszy modułu m
– jeśli moduł m dzieli się przez 4, to a - 1 również powinno być podzielne przez 4
Przykład generatora LCG (wg D. Knutha) $xi + 1 = (a * xi + c)modm$

x0 - ziarno
$c = [e * 109]$
m = 34359738368 = 235 e = 2,71828182845904523536028747135 27... - podstawa logarytmów naturalnych

1.3 Mersenne twister

Mersenne twister został opracowany przez Makoto Matsumoto i Takuji Nishimura w 1997 roku[2]. Generator ten dostarcza wysokiej jakości liczby pseudolosowe oraz jest bardo szybki. Nazwa pochodzi od tego, że na długość okresu została wybrana pierwsza liczba Mersenne'a. Algorytm, mimo swoich zalet, nie nadaje się do zastosowań kryptograficznych. Stosunkowo obserwacja niewielkiej liczby iteracji (624) pozwala przewidzieć wszystkie kolejne. Kolejną kwestią jest dlugi czas inicjalizacji algorytmu - w porównaniu do generatora Fibonacciego lub liniowego generatora kongruencyjnego.

1.4 Generator Park–Miller

Generator jest odmianą multiplikatywnego liniowego generatora kongruencyjnego, który określamy wzorem:

$$x_{n+1} = (a * x_n) \bmod M$$

gdzie, x_{n+1} to następna liczba pseudolosowa, a - współczynnik generujący kolejną liczbę, M - współczynnik określający zakres generowanych liczb (od 0 do $M - 1$).

```
1 private static final long max = ((long) 2 << 30) - 1;
2 private static final long a = 16807;
3
4 public static long getRandomNumber() {
5     seed = (a * seed) % max;
6     return seed;
7 }
```

Listing 4: Generowanie następnej liczby pseudolosowej przez algorytm Parka-Millera.

1.5 Xorshift

Algorytm został zaproponowany przez George Marsaglia[3] w 2003 roku. Liczba x_{n+1} jest generowana poprzez wielokrotną różnicę symetryczną liczb x_n i przesuniętej bitowo x_n. Wykorzystanie funkcji XOR sprawia, że ten

[2]http://doi.acm.org/10.1145/272991.272995
[3]http://www.jstatsoft.org/v08/i14/paper

algorytm jest niezwykle szybki na współczesnych komputerach.
Algorytm ten jest szybki ale nie niezawodny i z pewnością nie nadaje się do zastosowań kryptograficznych. Jednak, połączenie go z nieliniowym generatorem - jak pierwotnie sugerował autor - prowadzi do jednego z najszybszych generatorów, spełniających silne wymogi testów statystycznych.

```
public static long getRandomNumber() {
    seed ^= seed >> 12;
    seed ^= seed << 25;
    seed ^= seed >> 27;
    seed = (seed * 2685821657736338717L) % max;
    return Math.abs(seed);
}
```

Listing 5: Generowanie następnej liczby pseudolosowej przez Xorshift

Silnik przeglądarki internetowej Webkit korzysta z tego algorytmu przy wywoływaniu *Math.random()* w języku JavaScript.

Wnioski