Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1

An Algorithmic Random Integer Generator based on Prime Numbers Distribution

Bertrand Teguia Tabuguia

May 1, 2019

© 2019 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1

Abstract

More than a philosophic thinking, we combine two researchers wishes on randomness reproduction and prime numbers distribution. Indeed, up to now we cannot rigorously answer the question on randomness of primes [6, page 1]. We then propose an example of algorithms that can be deduced by that connection. For this purpose, our main procedure uses prime gap sequence variation. An evaluation on randomness reproduction is made at the end for a conclusion about prime numbers distribution and its implications.

Contents

Introduction **1**

1 Structure of Pseudo Random Integer Generator **2**

2 First Left Minimum (flm) Function **2**

3 Definition of Our Pseudo Random Integer Generator **3**

4 Application and Tests **5**

4.1 Probability variation of the prime gap sequence 7

4.2 χ^2- test of independence 8

Conclusion **10**

Introduction

The use of randomness is needed in almost all areas, some critical to mention are cryptography [4] and bioinformatics [1]. The latter article gives an alert about the use of built in random implementations. In general, we do not produce random since the approach is algorithmic, hence the common name pseudo random number generator (PRNG). PRNG's are all periodic, and obviously larger periods give better random imitation, so the issue about the period might sometimes be neglected. Another common issue with PRNG algorithms is the seeding or the initialization, since this is actually the construction of a sequence following states, two sequences having the same initial state must definitely be identical. Nevertheless a good seeding may result from the aim of the random implementation, for example in the P&C Game [7] where the random algorithm is actually the one presented here, the author uses the first click of the player to seed the generator.
We then come to the main core of a PRNG which is the black box to test for acknowledgment of pseudo randomness. Our algorithm is constructed to play that role. Positive integers having only two divisors, 1 or themselves, simple definition for big theories. These numbers

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1

called primes apart from belief up to now are not understood. And that is our concern, because randomness can be seen as a missing of information and so the question is why do not we use our ignorance to produce ignorance? This gives the importance of such a study because with enough improvement we may say that either we have produced random or understood primes distributions which turns out to be of great interest in research.
We have some interesting results and guess about primes seen as collection of numbers, we mention here the unsolved twin prime conjecture [8] which for prime gap might keep the randomization. Indeed this calls another interest in our algorithm because as we will see the twin prime conjecture has to be verified in order to keep all choices possible among the given labels.
In the sequel, after a brief presentation of a PRNG structure we will be giving details of our algorithm according to the given PRNG description. That is to define what we call first left minimum function (flm), and the update states. We end up with some tests for recognition of pseudo-randomness.

1 Structure of Pseudo Random Integer Generator

As basically presented in [2, chapter 3], typically there is a set S of states, and a function $f\colon S \longrightarrow S$ for state update. There is an output space O and function $h\colon S \longrightarrow O$. Usually the output space is taken to be the interval $(0,1)$, however in our case we will consider any finite set of labels or integers for simplicity. After choosing the seed, the sequence of random integers are generated as follows

$$\begin{aligned} S_n &= f(S_{n-1}), \quad n = 2,3,4,\dots \\ O_n &= h(S_n) \end{aligned} \tag{1}$$

2 First Left Minimum (flm) Function

Let $G = (g_1, g_2, \dots, g_n) \in \mathbb{E}^n$, where $\mathbb{E}$ is a non empty ordered set of objects. We define the first left minimum of G as

$$\operatorname{flm}_n(G) = \operatorname{flm}_n(g_1, g_2, \dots, g_n) = \begin{cases} g_1 & \text{if } g_1 < g_2 \\ g_2 & \text{if } g_2 < g_3 \text{ and } g_2 < g_1 \\ g_3 & \text{if } g_3 < g_4 \text{ and } g_3 < g_2 < g_1 \\ \dots & \\ g_{n-1} & \text{if } g_{n-1} < g_n \text{ and } g_{n-1} < g_{n-2} < \dots < g_1 \\ g_n & \text{if } g_n \leqslant g_{n-1} < g_{n-2} < \dots < g_1 \end{cases}, \tag{2}$$

indeed it is the first left minimum value between two consecutive components of G_n starting on the left.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1

Example: Let $\mathbb{E} = \mathbb{N}$, and consider $G_1 = (5,4,3,6,5), G_2 = (1,3,5,4,2)$, and $G_3 = (8,7,5,4,2)$. Then we have

$$\begin{aligned} \text{flm}_5(G_1) &= 3 \\ \text{flm}_5(G_2) &= 1 \\ \text{flm}_5(G_3) &= 2 \end{aligned} \tag{3}$$

From the definition of an flm function, one can easily deduce the following properties

Proposition 1. *Let $\mathbb{E}$ be a non empty ordered set. We have*

a)

$$\begin{aligned} \text{flm}_n \colon \mathbb{E}^n &\longrightarrow \mathbb{E} \\ G &\mapsto \text{flm}_n(G). \end{aligned}$$

b) $\text{flm}_2 = \min$.

c) Let $G = (g_1, g_2, \dots, g_n) \in \mathbb{E}^n$,

$$\begin{aligned} &\text{flm}_n(G) = \min\{g_1, g_2\}\,\delta_{g_1}(\min\{g_1, g_2\}) \\ &+ \delta_{g_2}(\min\{g_1, g_2\}) \Bigg(\min\{g_2, g_3\}\,\delta_{g_2}(\min\{g_2, g_3\}) + \delta_{g_3}(\min\{g_2, g_3\}) \Bigg(\dots \\ &+ \delta_{g_{n-1}}(\min\{g_{n-2}, g_{n-1}\}) \left(\min\{g_{n-1}, g_n\} \left(\delta_{g_{n-1}}(\min\{g_{n-1}, g_n\}) + \delta_{g_n}(\min\{g_{n-1}, g_n\})\right)\right) \dots \Bigg)\Bigg) \end{aligned} \tag{4}$$

where min *return the minimum value of its argument set and δ_g denotes the Kronecker symbol of g defined as*

$$\delta_g(v) = \begin{cases} 1 & \textit{if } v = g \\ 0 & \textit{otherwise} \end{cases}.$$

For c), if we take $G = (g_1, g_2, g_3)$ we have

$$flm_3(G) =$$

$$\min\{g_1, g_2\}\delta_{g_1}(\min\{g_1, g_2\}) + \delta_{g_2}(\min\{g_1, g_2\}) \left(\min\{g_2, g_3\} \left(\delta_{g_2}(\min\{g_2, g_3\}) + \delta_{g_3}(\min\{g_2, g_3\})\right)\right)$$

3 Definition of Our Pseudo Random Integer Generator

Instead of non empty as before, here we need an infinite countable ordered set. And for simplicity we consider $\mathbb{E} = \mathbb{N}$ since in any case $\mathbb{E}$ can always be assimilated to a subset of $\mathbb{N}$ by bijection. Given $N \geqslant 2$ integers $k_1, k_2, \dots, k_N$ for a random choice among them, we

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1

consider the set of states as a $N-$tuple of N consecutive prime gaps. If we denote by σ, a translation of the starting point in the primes set based on the seeding, then the set of state can be defined as

$$S = \left\{ S_n = (g_{n+1}, g_{n+2}, \ldots, g_{n+N}),\ g_j = p_{\sigma(j+1)} - p_{\sigma(j)},\ p_{\sigma(j)} \text{ primes},\ 1 \leqslant j - n \leqslant N, \right\}, \tag{5}$$

where n is a non negative integer.

The update state function is defined as the next ordered $N-$tuple of the prime gap sequence starting at $g_m = \text{flm}_N(S_n),\ n+1 \leqslant m \leqslant n+N$, that is

$$\begin{aligned} f\colon S &\longrightarrow S \\ S_n = (g_{n+1}, g_{n+2}, \ldots, g_{n+N}) &\longmapsto f(S_n) = S_{n+1} = (g_m, g_{m+1}, \ldots, g_{m+N-1}), \\ & \qquad g_m = \text{flm}_N(S_n),\ g_j = p_{\sigma(j+1)} - p_{\sigma(j)},\ 0 \leqslant j - m \leqslant N-1. \end{aligned} \tag{6}$$

Finally the output function is given by

$$\begin{aligned} h\colon S &\longrightarrow \{k_1, k_2, \ldots, k_N\} \\ S_n &\longmapsto h(S_n) = k_{\text{indexOf}(\text{flm}_N(S_n), S_n)}, \end{aligned} \tag{7}$$

where $\text{indexOf}(g_j, S_n)$ returns the index of g_j in the tuple S_n counted from the left.

Remark: The seed impact is an important aspect to highlight, because it tells us how σ is chose. Indeed if for example we have $\sigma(j) = j+4$, then $g_1 = p_{\sigma(2)} - p_{\sigma(1)} = p_6 - p_5 = 13 - 11 = 2$ and $g_2 = 17 - 13 = 4$. In such a case for $N = 2$ we obtain k_1 as the first output. Thus depending on the situation, one has to define his own σ from the seed.

However, as the goodness of the algorithm also depends on the period, a question to answer is the one related to an estimation of the period. This of course rely on the prime number theorem which states the following

Theorem 3.1 (Prime Number Theorem (see [6])). *The number of primes less than a given integer n is*

$$(1+\varepsilon_n)\frac{n}{\ln n}, \quad \lim_{n\to\infty} \varepsilon_n = 0. \tag{8}$$

Where $\ln$ *denotes the natural logarithm.*

Therefore given the maximal integer reachable by the working programming language or software one can estimate the period of our PRNG algorithm using the prime number theorem. Having such an estimation, extremity (max and min) behavior has to be defined to make sure the algorithm continues, basically the need is to define the gap between the maximum prime and 2.

Nevertheless there are infinitely many primes, so as far as the system can go in the calculation of large prime numbers as big will be the period of the algorithm. But in revenge, the apparent concern will be the speed that could reduce the largest possible period to the largest accessible in a short time.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1

Remark:

- If the twin primes conjecture is verified, then we are sure that our algorithm will always be able to change the component choice in a state S_n since twin primes give the smallest prime gap for high integers.
- More generally, any unstable behavior of primes can be used in this way to imitate random. The gap variation is just an example since we cannot be sure on its increasing, decreasing or constant behavior.

Next, we evaluate our algorithm. This may give us a probabilistic argument about primes distribution.

4 Application and Tests

For tests, we consider the two labels case. This corresponds to Carole's behavior in the P&C Game for the Prime level (see [7]). We are going to generate a sequence of sequence of $0, 1$ following our pseudo random algorithm and make a test on independence and uniformity as explain in [2, chapter 3]. Indeed let n, d be two large enough integers. We generate n $d-$tuple of $\{0, 1\}$ by our algorithm and check uniformly distribution with the χ^2-test.
For application we seed the generator with the sth prime number as the minimal prime of the initial gap, where s is taken randomly on a certain interval in the system used. For this purpose, simple codes to generate a csv test file, can be written in python 2.7 [3] as follows

- random code:

```
#BTrandom2.py file
#By Bertrand Teguia
#Random code for two labels (0,1) based on prime numbers distribution

from math import *
import random


def nextprime(p):
    if p > 2:
        value = p
        while True:
            i = 3
            value += 2
            q = int(floor(sqrt(value)))
            while i <= q and value % i:
                i += 2
            if i > q:
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1


```
                break
            return value
        value = 3 if p==2 else 2
        return value



    def nthprime(n):
        cpt=1
        p=2
        while cpt<n:
            p=nextprime(p)
            cpt+=1
        return p



    def BTrandom2():
        global prime
        global gap
        prevprime = prime
        prime = nextprime(prime)
        prevgap = gap
        gap = prime - prevprime;
        if prevgap < gap:
            return 0
        else:
            return 1

    # seeding
    s = nthprime(random.randint(10000, 20000))
    prime = nextprime(s)
    gap = prime - s
```

- csv file creator code:

```
#BTrandom2_test.py file
#By Bertrand Teguia
#Random code csv generator of n d-tuple for BTrandom2


from BTrandom2 import *

import csv
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1


```
d = 100
n = 100

lines = []
for i in range(n):
    row = []
    for j in range(d):
        row.append(BTrandom2())
    lines.append(row)

with open('BTrandom2_test.csv', 'w') as writeFile:
    writer = csv.writer(writeFile)
    writer.writerows(lines)

writeFile.close()
```

Running the file *BTrandom2_ test.py* in the same folder with *BTrandom2.py* produces a csv file named *BTrandom2_ test.csv*. We can thus generate many files as we want for experiences.

4.1 Probability variation of the prime gap sequence

Before going through the χ^2-test, let us first give an estimation of the probability p that 1 appears. Thus we use the law of large numbers [5], so we consider the outcomes from our generator to be independent. As seen in our code, especially for this part we consider $n = d = 100$. The law of large number tells us that an estimation of the expectation of the distribution follow by our generator, which is the expectation of distribution followed by the prime gap increasing behavior at any index, is the limit

$$E = \lim_{d \longrightarrow \infty} \frac{\sum_{j=1}^{d} x_j}{d}, \tag{9}$$

where x_j, $1 \leqslant j \leqslant d$ denote the observations of the trial processes of a row in our generated csv files. After many computations from csv files we realized that the expectation oscillates between 0.48 and 0.57, and moreover using again the law for the estimated expectations on each raw gives us amazingly almost exactly in all the cases the value 0.52.

Note that the second use of the law of large numbers adds the hypothesis that each row are taken independently which is natural from the first independence hypothesis. Furthermore, due to the experience characteristics, one trivially sees that we are in the case of a Bernoulli picture. Therefore since the expectation of a Bernoulli experience is the winning probability we have our estimation, that is to say that the probability that the prime gap sequence increases at any index is given by

$$p = E = 0.52, \tag{10}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1

or rigorously, in the convenient probability space with the probability $\mathcal{P}$, given any three consecutive prime numbers

$$p_n < p_{n+1} < p_{n+2},\ n \in \mathbb{N},$$

we have

$$\mathcal{P}\left\{p_{n+2} - p_{n+1} \geqslant p_{n+1} - p_n\right\} = 0.52. \tag{11}$$

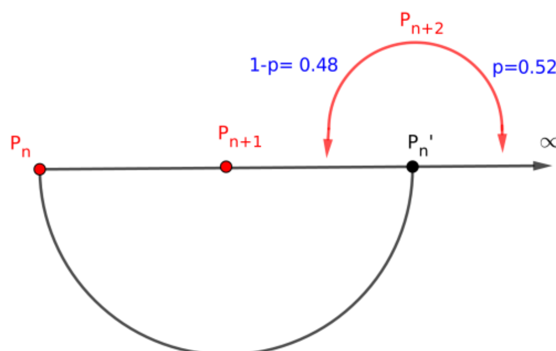

Figure 1: Prime numbers occurrence

By the closure of the value obtained with 0.5 (uniformity), from this result one might say that the variation of the prime gap sequence is random. So we may think at this stage that our random integer generator produces a good imitation of randomness. Note however that talking about random for a sequence of numbers, is more for the unknowns or not computed values of that sequence when going to infinity. Indeed there is no random to expect from value that we already have. Though it might be a shock for us mathematicians, one should rigorously talk this way. A picture of this result is given in Figure 1.

Notice that, using the χ^2-test for individual variation as we have just done will lead to the same conclusion. In the next paragraph, we give the result from the χ^2- test on a different view of the prime gap sequence.

4.2 χ^2- test of independence

We are going to check if the row in our generated csv files are uniformly distributed over $\{0, 1\}^d$, we then choose $d = 4$ and $n = 10000$. This tests the independence to some extends but it only tests if d consecutive calls of our generator are independent. Note however that the code producing the csv file has to change a little bit, because we prefer to count occurrences directly in python. The new code looks as follow

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1


```
#BTrandom2_testX2.py file
#By Bertrand Teguia
#Random code csv generator for a X^2 test of BTrandom2


from BTrandom2 import *

import csv

d = 2
n = 20000

lines = []
for i in range(n):
    row = []
    for j in range(d):
        row.append(BTrandom2())
    lines.append(''.join(map(str, row)))

lines = map(lambda y: y.split(),list(set(map(lambda x:\
    x+" "+str(lines.count(x)), lines))))

with open('BTrandom2_testX2.csv', 'w') as writeFile:
    writer = csv.writer(writeFile)
    writer.writerows(lines)

writeFile.close()
```

Running this update python code lead us to the following table 1, where $E_i = np_i$, with $p_i = \frac{1}{16}$ denote the expectation of O_i : occurrence number of the label i. o_i is the observed value. Remember that our null hypothesis is to have uniform distribution ($p_i = \frac{1}{16}$, $\forall i$).

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1

Table 1: χ^2-test of independence for 4 consecutive variations of the prime gap sequence

i	O_i	E_i	$\frac{(O_i-E_i)^2}{E_i}$	
1111	104	625	434,3056	
0000	49	625	530,8416	
0011	501	625	24,6016	
0111	399	625	81,7216	
1010	1326	625	786,2416	
1101	818	625	59,5984	
1100	490	625	29,16	
1000	275	625	196	
0010	676	625	4,1616	
1001	859	625	87,6096	
1011	835	625	70,56	
1110	422	625	65,9344	
0110	944	625	162,8176	
0001	236	625	242,1136	
0101	1414	625	996,0336	
0100	652	625	1,1664	
	10000	10000	3772.8672	Total

As we have $16 - 1 = 15$ degrees of freedom, from the χ^2-distribution table one sees that we are really far away from uniformity, so we reject our null hypothesis. Thus despite the random behavior observed previously, the prime gap sequence rather appears to not behave randomly when you consider collection of its different variations.

Conclusion

Terence Tao concluded «*Individual primes are believed to behave randomly, but the collective behavior of the primes is believed to be quite predictable*»[6]. Here we have find out an argument to improve that statement and rather say that consecutive primes appear randomly but taken as small groups the argument of randomness for primes is rejected. Thus our random imitation from prime numbers is actually not a good random. Nevertheless, for its use in the P&C Game [7], the author is right on using this algorithm. Indeed, as seen in the previous section, a consecutive constant behavior of the prime gap sequence variation does not happen often. That means the average displacement of a player during a game play is less in general.

For the study of primes occurrence, we think that a good recommendation for further understanding on their behavior is to consider them on small groups or better small groups of consecutive primes.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019
doi:10.20944/preprints201905.0036.v1

Furthermore, as coming from a natural phenomenon, here prime numbers appearance, our algorithm might produce interesting study subject in term of precess.

Acknowledgments: I would like to thank AIMS-Cameroon for the facility that they give to do research.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 6 May 2019 doi:10.20944/preprints201905.0036.v1

References

[1] David Jones. Good practice in (pseudo) random number generation for bioinformatics applications. *URL http://www.cs.ucl.ac.uk/staff/d.jones/GoodPracticeRNG.pdf*, 2010.

[2] Tom Kennedy. *Monte Carlo Methods - a special topics course*. University of Arizona, 2016.

[3] Python Software Foundation. Python 2.7.0 release. 2019.

[4] OF TRUE RANDOMNESS. The importance of true randomness in cryptography.

[5] John Renze and Eric W. Weisstein. Law of large numbers.

[6] Terence Tao. Structure and randomness in the prime numbers. In *An Invitation to Mathematics*, pages 1–7. Springer, 2011.

[7] Bertrand Teguia T. P&C Game. 2019.

[8] Eric W Weisstein. Twin primes. 2003.