A pseudo-random generator whose output is a normal sequence

Boris Ryabko[1,2]

[1]Federal Research Center for Information and Computational Technologies,
[2]Novosibirsk State University, Novosibirsk, Russia
boris@ryabko.net

Abstract

Pseudo-random number generators (PRNGs) are widely used in computer simulation, cryptography, and many other fields. In this paper, we describe a PRNG class, which, firstly, has been successfully tested using the most powerful modern test batteries, and secondly, is proved to consist of generators that generate normal sequences. The latter property means that, for any generated sequence $x_1x_2...$ and any binary word w, we have

$$\lim_{t\to\infty} \nu_t(w)/(t-|w|) = 2^{-|w|},$$

where $\nu_t(w)$ is the number of occurrences of w in the sequence $x_1...x_{|w|}$, $x_2...x_{|w|+1}$, ..., $x_{t-|w|+1}...x_t$.

Keywords: pseudo-random number generators, normal sequences, randomness testing.

1 Introduction

Pseudo-random number generators (PRNG) are designed to generate sequences of binary digits, which are distributed as a result of throwing a "fair" coin, or, to be precise, obey the Bernoulli distribution with the parameters $(1/2, 1/2)$. This i.i.d. process and sequences generated by it are often called "truly random" [1, 2].

Truly random sequences are very desirable in simulation, cryptography, and modeling applications. Of course, it is too slow to generate them tossing a coin, and nowadays there are many so-called generators of pseudo-random numbers, whose aim is, informally speaking, to construct sequences that mimic the truly random (see [1, 2]).

A modern PRNG is a computer program with an input word ("seed") and an output binary words and the length of output is significantly grater

(in bits). Considering that the seed is a true random word, PRNG can be thought of as a randomness expander that expands the seed into a longer word [1, 2].

The RNG theory is very close in spirit to the concept of random sequence and randomness in general. But it turns out that one of the consequences of the concept of random sequence shows that PRNG does not exist. More precisely, a mathematically correct definition of a random sequence was obtained in a framework of algorithmic information theory established by Kolmogorov (see [3, 4, 5, 6, 7, 8]). In particular, it is shown that any algorithm (i.e., a Turing machine) can neither generate (infinite) random sequences nor stretch a short random sequence into a longer one. It means that PRNGs do not exist. The same is true in the framework of Shannon's information theory. Indeed, it is known that the Shannon entropy of the true random process is one bit per letter, whereas for all other binary processes the entropy (per letter) is less than one (see [9]). On the other hand, any PRNG stretches a short true random sequence into a long one. So, the entropy of the output is not greater than the entropy of the input and, hence, the per letter entropy of the output is strictly less than 1 bit. Therefore, the demands of true randomness and low entropy are contradictory. (On the other hand, there are different approaches to defining a "random sequence", and the existence of an ideal PRNG does not contradict some of them, see [10]).

Thus, we see that, in the framework of algorithmic information theory, as well as in the framework of Shannon information theory, "perfect" PRNGs do not exist.

In such a situation, researchers suggest and investigate PRNGs that meet some "probabilistic" properties of truly random sequences [1, 2, 11]. One of such properties is that a PRNG generates so-called normal (or ∞-distributed) sequences. The following definition of normal sequences belongs to Borel [12]: A sequence of digits in base 2 is k-distributed if for any k-letter word w over the alphabet $\{0, 1\}$

$$\lim_{t\to\infty} \nu_t(w)/(t-k) = 2^{-k} \tag{1}$$

where $\nu_t(w)$ is a number of occurrences of w in the sequence $x_1...x_k$, $x_2...x_{k+1}, ..., x_{t-k+1}...x_t$. The sequence is normal (or ∞-distributed) if it is k-distributed for any $k \geq 1$. (Borel called normal a real number from the interval $(0, 1)$ whose expansion in base 2 is a normal sequence, and showed that almost all real numbers are normal (with respect to the uniform measure) [12]. That is why, the property that PRNG generates normal sequences is very desirable.

In [13], the so-called two-faced processes were described, which, on the one hand, generate normal sequences, and on the other hand, their entropy (per letter) may be close to zero. Thus, in principle, they can be considered as a promising tool for constructing PRNGs, since their entropy per letter is small, but the output is a normal sequence. The only problem is that the property of being a normal sequence is asymptotic, but, informally, real PRNGs must generate sequences that are recognized by random modern battery of tests for any length of the sequence. In short, we call such PRNGs statistically acceptable.

This paper aims to suggest such PRNGs that

– generate normal sequences.

– generate statistically acceptable sequences of any length.

The first property was proved mathematically in [13], and the second one was tested experimentally using batteries of statistical tests [14, 15, 16], which are currently considered the most powerful.

We refer to the generator proposed here as the PRNG with asymptotically proven statistical properties (or PRNG APSP).

The rest of the paper is as follows: the next part describes two-faced random processes and PRNGs that generate normal sequences. The third part contains the results of experiments which show that proposed PRNGs generate sequences which are recognized as truly random by the batteries of tests from [14, 15, 16].

2 Two-faced processes and PRNG generated normal sequences

2.1 Two-faced Markov chains

The purpose of this part is an informal explanation of the basic ideas underlying the proposed PRNG, which are connected with so-called two-faced Markov chains.

First we consider several examples of two-faced Markov chains. Let a matrix of transition probabilities T_1 be as follows:

$$T_1 = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & \nu & 1-\nu \\ 1 & 1-\nu & \nu \end{array} , \tag{2}$$

where $\nu \in (0,1)$ (i.e. $P\{x_{i+1}=0|x_i=0\}=\nu$, $P\{x_{i+1}=0|x_i=1\}=1-\nu, ...$).

For example, let $\nu = 0.9$. Then, a "typical" output sequence can be as follows:

$$111111111\ \ 0000000000\ \ 1111111\ \ 00000000000\ \ ...$$

(Here the gaps correspond to state transitions.) If $\nu = 0.1$, then a "typical" output sequence is

$$101010101\ \ 1010101010\ \ 0101010101010101\ \ 1010\,...\ .$$

On the one hand, these sequences are not truly random. On the other hand, the frequencies of 1's and 0's go to 1/2 due to the symmetry of the matrix (2). Hence, the output is 1-distributed.

It is shown in [13] that the following Markov chain with memory 2 generates 2-distributed random sequences:

$$T_2 = \begin{array}{c|cccc} & 00 & 01 & 10 & 11 \\ \hline 0 & \nu & 1-\nu & 1-\nu & \nu \\ 1 & 1-\nu & \nu & \nu & 1-\nu \end{array}$$

(Here $P\{x_{i+1}=0|x_i=0,x_{i-1}=0\}=\nu$, $P\{x_{i+1}=0|x_i=0,x_{i-1}=1\}=1-\nu, ...$). For $\nu = 0.9$ the "typical" output sequence can be as follows:

$$000000000000\ \ 1101101101101101101101101110\ \ 000\,...$$

It can be easily seen that the frequency of any two-letter word goes to 1/4.

This simple example explains the expression "two-faced process". The output of the sequence mimics true random sequences up to a word length of 2, but for lengths greater than 2 the output is "absolutely" non-random.

We do not give the definition of Markov chains generated by k-distributed sequences, because they are not used directly in the proposed PRNG, but mentioned them, since they informally describe the main idea of the PRNG construction. More precisely, the existence of Markov chains generating (with probability one) k -distributed sequences shows that for any integer k there are random processes with low entropy, whose outputs simulate true random sequences up to the word length k. It is important that the described Markov chains are k -distributed due to the symmetry, therefore all words of length k have the same probabilities.

2.2 ∞-distributed processes

First, we give a formal definition of the two-faced processes.

Definition. Let there be a random process generating binary sequences. The process is two-faced of order k if the equation

$$P(x_{j+1}...x_{j+k} = u) = 2^{-|u|}. \tag{3}$$

is true for any u of the length k.

The process is asymptotically two-faced of order k if

$$\lim_{j\to\infty} P(x_{j+1}...x_{j+k} = u) = 2^{-|u|}. \tag{4}$$

The following simple statement from [13] shows that, in a certain sense, there are many two-faced processes. More precisely, the following theorem is true.

Theorem 1. *Let $X = x_1x_2...$, $Y = y_1y_2...$ be an independent random processes. We define the process $Z = z_1z_2...$ by the following equations $z_1 = x_1 \oplus y_1$, $z_2 = x_2 \oplus y_2, ...$ where $a \oplus b = (a + b) \, mod \, 2$. If X is a k-order (asymptotically) two-faced process, then Z is also the k-order (asymptotically) two-faced process ($k \geq 1$).*

Now, we describe a family of processes that generate ∞- distributed sequences. Suppose that $m^* = m_1, m_2, ...$ is a sequence of integers, $m_1 < m_2 < m_3....$ and $X^1 = x_1^1x_2^1...$, $X^2 = x_1^2x_2^2...$, $X^3 = x_1^3x_2^3...$, ... are (asymptotically) two-faced processes of order $m_1, m_2, ...$, correspondingly. Define a process $W = w_1w_2$ ... by the following equation:

$$w_i = \begin{cases} x_i^1 & i \leq m_1, \\ x_i^1 \oplus x_i^2 & m_1 < i \leq m_2, \\ x_i^1 \oplus x_i^2 \oplus x_i^3 & m_2 < i \leq m_3, \\ \quad ..... & \end{cases} \tag{5}$$

and denote it as $\sum_{i=1}^{\infty} X^i$.

Theorem 2. *[13] Let X^i, $i = 1, 2, ...$, be asymptotically two-faced of order $m_1, m_2, ...$, correspondingly, and $m_1 < m_2 < m_3....$. Then $\sum_{i=1}^{\infty} X^i$ is normal (with probability one).*

Note that the proof immediately follows from Theorem 1.

The proposed PRNG construction repeats the process W in (5), but before describing it, we need to describe how to transform any random process into k -distributed for any integer k. It can be done as follows:

Theorem 3. *[13] Let there be a stationary ergodic source S generating a sequence $u_1u_2...$ and a uniformly distributed word $x_{-k+1}x_{-k+2}...x_0$, $k \geq 1$. Then S can be transformed into k-distributed source X as follows:*

$$x_n = u_{n+1} \oplus \bigoplus_{i=n-k}^{n-1} x_i, \;\; n = 1, 2, ... \; . \tag{6}$$

If the word $x_{-k+1}x_{-k+2}...x_0$ is generated according to some non-uniform distribution and the Shannon entropy of the source S is greater than zero, then X is asymptotically k-distributed.

2.3 Description of the PRNG

Let's first informally describe the proposed algorithm, which is based on the above statements. Consider a sequence where each term is equal to the XOR of k previous terms (for some fixed k) and imagine that we follow this recurrent rule (XOR of k previous terms) most of the time, but from time to time use the opposite bit (the moments when this happens are determined by the seed). The resulting sequence can be used as a pseudorandom candidate. To improve its "pseudorandomness", one could generate several sequences of this type, for different k (preferably prime) and apply final XOR to them.

More formally, the proposed algorithm is an implementation of (5) and (6). The proposed PRNG generates a sequence $x_1^{output}x_2^{output}....x_N^{output}$ of the required length N and this sequence is the sum of s two-faced processes $\hat{x}^1 = x_1^1...x_N^1$, $\hat{x}^2 = x_1^2...x_N^2$, , ..., $\hat{x}^s = x_1^s...x_N^s$ of orders $m_1, m_2, ..., m_s$, that is, $x_i^{output} = \bigoplus_{n=1}^{s} x_i^n$, $i = 1, ..., N$ (here parameters s and $m_1 < m_2 < ...$ $< m_s$ can depend on N). Let $\hat{r} = r_1...r_R$ be a seed, that is, the sequence of R random bits, where

$$R = m_1 + \lceil \log_2 m_1 \rceil + \sum_{i=1}^{s} \lceil N/m_i \rceil. \tag{7}$$

All sequences $\hat{x}^i$ are generated according to transformation (6) and, hence, we first should describe corresponding sequences $\hat{u}_i, i = 1, ..., s$. For this we will use the following auxiliary notation: for a binary word z, we denote by $int(z)$ an integer whose binary representation is equal to z. So, the sequence $\hat{u}^1$ is defined as follows:

$$u^1_{i+\delta} = \begin{cases} 0, & if \;\; (i-1)/m_1 \;\; is \; not \; integer \\ r_{\lceil \log_2 m_1 \rceil + m_1 + i/m_1}, & if \;\; (i-1)/m_1 \;\; is \; integer \end{cases},$$

where $\delta = int(r_1 r_2 ... r_{\lceil \log_2 m_1 \rceil})$, $i = 1, ..., N$. The sequence $\hat{x}^1$ is defined by (6)

$$x_n^1 = u_{n+1}^1 \oplus \bigoplus_{i=n-m_1}^{n-1} x_i^1, \;\; n = 1, 2, ... \; .$$

where $x_{-m_1+1}^1 ... x_0^1 = r_{\lceil \log_2 m_1 \rceil + 1} ... r_{\lceil \log_2 m_1 \rceil + m_1}$.

In short, we read $\lceil \log_2 m_1 \rceil$ first random bits and find $\delta = int(r_1 r_2 ... r_{\lceil \log_2 m_1 \rceil})$. Then the letters $u_\delta^1, u_{\delta+m_1}^2, u_{\delta+2m_1}^3, ...$ are chosen randomly, that is, they are $r_{m_1+\lceil \log_2 m_1 \rceil+1}$, $r_{m_1+\lceil \log_2 m_1 \rceil+2}$, $r_{m_1+\lceil \log_2 m_1 \rceil+3}, ...$, whereas all other u_i^1 are 0.

It is important that a random choice of the "shift" δ makes $u_1^1, u_1^2, ...$ stationary ergodic. Hence, we can apply the theorem 3. On the other hand, the main idea of transformation (as well as the PRNG) can be briefly informally explained as follows: the output is k -distributed due to the symmetry of the sequence transformations (as for two-faced Markov chains).

Processes $\hat{x}^j, j = 2, ..., s$, are defined analogically:

$$u_{i+\delta_j}^j = \begin{cases} 0, & if \quad i < m_j \, , \\ 0, & if \quad (i-1)/m_j \;\; is \; not \; integer \\ r_{\lceil \log_2 m_1 \rceil + \sum_{n=1}^{j-1} \lceil N/m_n \rceil + i/m_j}, & if \quad (i-1)/m_j \;\; is \; integer \end{cases} ,$$

$$x_{-m_j+1}^j x_{-m_j+2}^j ... x_0^j = x_1^{output} ... x_{m_j}^{output}, \;\; x_n^j = u_{n+1}^j \oplus \bigoplus_{i=n-m_j}^{n-1} x_i^j, n = 1, 2, ..., N.$$

(Note that $x_1^{output} ... x_{m_j}^{output}$ are defined when $x_{-m_j+1}^j x_{-m_j+2}^j ... x_0^j$ are calculated and all calculations can be performed in such a way that the sequences x_i^1, $..., x_i^s$, $i = 1, ..., N$, are not stored in memory).

Let us give some comments about a practical realisation. An important issue is the length of the seed. The length R in (7) depends on the length N of the output sequence. Generally speaking, this size can be reduced if we take into account the second statement of the Theorem 3. More precisely, suppose there is a random word r that can be used as a seed, and $|r| < R$. In this case, we propose to first generate a sequence of R -bits using the described PRNG, and then use it as a seed to generate an output of length N.

In addition, we suggest using prime numbers m_i to increase "mixing".

3 Experimental investigation of the proposed PRNG

In this part we describe the results of experiments that were carried out with the PRNG introduced above (Its implementation can be found there [17, 18].). We tested the output of the PRNG by tests from the well-known test batteries suggested in [14, 15, 16].

In our experiments, we used the PRNG under consideration with three parameters m_1, m_2, m_3, which were investigated for different values and different lengths of the output sequence.

It is important to note that in all the experiments described below, the seed was obtained in two ways: first, the PRNG MRG32k3a from [14, 10] was used to obtain R bits (see (7)), and, second, the length of the "real" seed was 256 bits generated by MRG32k3a, and, then the described PRNG APSP generated the required R bits with the parameters $m_1 = 127$, $m_2 = 907$, which, in turn, were used as the seed for the PRNG APSP with other parameters. In the examples, the results for both versions of the seed were the same.

The results of the experiments are presented in Table 1. When using batteries Alphabit and Rabbit the length of the output sequence was 1 GB, while other batteries define their strategy for using output sequences themselves. When using the NIST battery [15], the length of the output sequence was 100 MB (1 GB cannot be used for this test).

The experiments show that there are values of the parameters m_1, m_2 for which the output sequences cannot be distinguished from truly random ones using the best modern battery tests. In particular, the PRNG with parameters $m_1 = 1021$, $m_2 = 1021001$, $m_1 = 1021$, $m_2 = 102101$ and some others can be recommended for practical use.

We see that statistically acceptable PRNGs can be obtained with m_1 and m_2. Similar experiments were carried out with different parameters m_1, m_2 and m_3. Based on this experiments we recommend the following parameters

$$m_1 = 127, m_2 = 12703, m_3 = 1021001,$$

$$m_1 = 257, m_2 = 25703, m_3 = 9834497 \qquad .$$

We did not carry out experiments with 4 and more parameters m_i, but based on the asymptotic results (Theorems 1 and 2) we can recommend to use such prime numbers that $m_i > m_{i-1}m_{i-2}$ for $i > 3$. For example, if the length of the output sequence is larger than 1 terabyte, it would make sense to use $m_4 = 4398042316799$.

Table 1: Results of the experiments.

m_1	m_2	NIST	Alphabit	Rabbit	SmallCrush	Crush	BigCrush	PractRand
31	257	Passed	Passed	Rejected	Passed	Rejected	Rejected	Rejected
31	907	Passed	Passed	Passed	Passed	Passed	Rejected	Rejected
31	3571	Passed	Passed	Passed	Passed	Passed	Passed	Rejected
61	257	Passed	Passed	Rejected	Passed	Rejected	Rejected	Rejected
61	907	Passed	Passed	Passed	Passed	Passed	Passed	Passed
61	3571	Passed	Passed	Passed	Passed	Passed	Passed	Passed
127	257	Passed	Passed	Passed	Passed	Rejected	Rejected	Rejected
127	907	Passed	Passed	Passed	Passed	Passed	Passed	Passed
127	3571	Passed	Passed	Passed	Passed	Passed	Passed	Passed
127	1277	Passed	Passed	Passed	Passed	Passed	Passed	Passed
127	12703	Passed	Passed	Passed	Passed	Passed	Passed	Passed
127	126997	Passed	Passed	Passed	Passed	Passed	Passed	Passed
257	2579	Passed	Passed	Passed	Passed	Passed	Passed	Passed
257	25703	Passed	Passed	Passed	Passed	Passed	Passed	Passed
257	257003	Passed	Passed	Passed	Passed	Passed	Passed	Passed
509	5087	Passed	Passed	Passed	Passed	Passed	Passed	Passed
509	50893	Passed	Passed	Passed	Passed	Passed	Passed	Passed
509	508987	Passed	Passed	Passed	Passed	Passed	Passed	Passed
1021	10211	Passed	Passed	Passed	Passed	Passed	Passed	Passed
1021	102101	Passed	Passed	Passed	Passed	Passed	Passed	Passed
1021	1021001	Passed	Passed	Passed	Passed	Passed	Passed	Passed

4 Conclusion

In this article, we proposed a class of PRNG APSP that generate normal sequences (as far as the asymptotic properties can be valid for a real computer program). Within this generator, several m_i -distributed sequences are generated for several different values of m_i, and then an output sequence is generated by summing these m_i -distributed sequences. This property is based on a symmetric mixing transformation.

A series of experiments made it possible to find PRNG parameters for which the PRNG has passed well-known batteries of statistical tests. Thus, the proposed PRNGs combine an acceptable practical behavior with mathematically proven asymptotic properties.

This PRNG can be interesting for practical use as such and, due to the property of normality, can be used to improve other RNGs. For example, if a certain RNG generates sequences $x_1, x_2, ...$ that satisfy some special properties (say, important for cryptography), and the PRNG APSP generates the sequence $y_1, y_2, ...$, then the sequence $x_1 \oplus y_1$, $x_2 \oplus y_2, ...$ will be normal and usually retain the special property.

Acknowledgment

The author is grateful to V. Zhuravlev, who implemented the described PRNG and carried out experiments.

This work was supported by the Russian Foundation for Basic Research (grant 18-29-03005).

References

[1] L'Ecuyer, P. History of uniform random number generation. In Proceedings of the WSC 2017-Winter Simulation Conference, Las Vegas, NV, USA, 3–6 December 2017.

[2] P. L'Ecuyer. Random number generation. In J. E. Gentle, W. Haerdle, and Y. Mori, ed-itors, Handbook of Computational Statistics, pages 35–71. Springer-Verlag, Berlin, second edition, 2012.

[3] Li, M.; Vitányi, P. *An Introduction to Kolmogorov Complexity and Its Applications*; Springer-Verlag: Berlin/Heidelberg, Germany, 2008.

[4] Calude, C.S. *Information and Randomness—An Algorithmic Perspective*; Springer-Verlag: Berlin/Heidelberg, Germany, 2002.

[5] Downey, R.G.; Hirschfeldt, D.R. *Algorithmic Randomness and Complexity*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2010.

[6] Downey, R.; Hirschfeldt, D.R.; Nies, A.; Terwijn, S.A. Calibrating randomness. *Bull. Symb. Log.* **2006**, *12*, 411–491.

[7] Merkle, W.; Miller, J.S.; Nies, A.; Reimann, J.; Stephan, F. Kolmogorov–loveland randomness and stochasticity. *Ann. Pure Appl. Log.* **2006**, *138*, 183–210.

[8] Durand B., Kanovei V., Uspensky V., Vereshchagin N. Do stronger definitions of randomness exist? *Theoretical Computer Science* 290:3 (2003) 1987-1996.

[9] Cover, T.M.; Thomas, J.A. *Elements of Information Theory*; Wiley-Interscience: New York, NY, USA, 2006.

[10] P. L'Ecuyer. Good parameters and implementations for combined multiple recursive random number generators. Operations Research, 47(1):159–164, 1999.

[11] L'Ecuyer, P. *Random Number Generation and Quasi-Monte Carlo*; Wiley: Hoboken, NJ, USA, 2014.

[12] Borel, E. Le continu mathématique et le continu physique. 1909.

[13] B. Ryabko, Low-Entropy Stochastic Processes for Generating k-Distributed and Normal Sequences, and the Relationship of These Processes with Random Number Generators. *Mathematics* , 7(9), 838, 2019.

[14] L'Ecuyer, P.; Simard, R. TestU01: A C library for empirical testing of random number generators. *ACM Trans. Math. Softw. (TOMS)* **2007**, *33*, 22.

[15] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, *A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications.* National Institute of Standards and Technology, 2010.

[16] PractRand. http://pracrand.sourceforge.net/

[17] https://github.com/zhuravlev12/Generator (01.12.2020).

[18] https://github.com/zhuravlev12/Generator2.0 (01.12.2020).