STATISTICAL ANALYSIS OF RANDOM NUMBER GENERATORS

LUIGI ACCARDI

Università di Roma Tor Vergata, Centro Interdipartimentale Vito Volterra, Via Columbia 2, 00133 Roma, Italy, E-Mail: accardi@volterra.uniroma2.it

MARKUS GÄBLER

Brandenburg Technical University Cottbus, Department of Mathematics, PO box 101344, 03013 Cottbus, Germany, E-Mail: gaebler@math.tu-cottbus.de

In many applications, for example cryptography and Monte Carlo simulation, there is need for random numbers. Any procedure, algorithm or device which is intended to produce such is called a random number generator (RNG). What makes a good RNG? This paper gives an overview on empirical testing of the statistical properties of the sequences produced by RNGs and special software packages designed for that purpose. We also present the results of applying a particular test suite—TestU01—to a family of RNGs currently being developed at the Centro Interdipartimentale Vito Volterra (CIVV), Roma, Italy.

1. Introduction

Assume $X_1, \ldots, X_N$ is a random binary sequence, i.e. for all $1 \leq k \leq N$ X_k is a Bernoulli random variable with $P(X_k = 1) = p_k \in (0, 1)$. Such a sequence is called purely random, if $X_1, \ldots, X_N$ are independent and identically distributed (i.i.d.) with $p_k = p = 1/2$.

In many applications, for example cryptography and Monte Carlo simulation, there is need for random numbers. Using binary expansion and transformation methods any such random numbers can be constructed from purely random binary sequences to an arbitrary precision, ignoring numerical difficulties for the moment. So many ways have been invented to produce, or at least simulate, realizations $x_1, \ldots, x_N$ of such sequences. Repeated flipping of a "fair" coin and recording "0" for "heads" and "1" for "tails" would be one way, but surely too timeconsuming, if you needed, say, 10^{18} exponential random numbers for some extensive stochastic simulation. Using randomness in physical quantities like noise in an electrical circuit or the timing of strokes at a user keyboard would be another. Or even manipulation of a definitely deterministic process (like a small computer programm) in a way that the outcomes appear to be purely random might serve just as well. Any such procedure, algorithm or device which is intended to produce realizations of random sequences, we call a random number generator (RNG). Deterministic RNGs are also sometimes called pseudo-random number generators (PRNG) or algorithmic RNGs. But no matter what the nature of a particular RNG is, the question arises whether the sequences it produces can be distinguished from the ones coming from the purely random theoretical ideal.

This paper only deals with the **statistical properties** of binary sequences (viewed as realizations of random binary sequences). But while good statistical properties are necessary, there are, of course, other important quality criteria, including:

Efficiency with respect to time and memory.

Sufficiently large period: The internal state of a deterministic RNG runs over a finite set and is therefore periodic, so running through a significant portion of the whole cycle should be beyond reach in practice.

Repeatability: the ability to reproduce the same sequence as many times as needed. Non-deterministic RNGs are not repeatable.

Portability: independence from software and hardware environment.

Unpredictability: The next output bit of an RNG is not to be predicted from knowing preceding ones any better than by “tossing a fair coin”. For a deterministic RNG this means that it should not be possible to find the internal state and/or the transition law from knowing the output sequence, at least not in reasonable time employing a reasonable amount of resources.

The importance of these criteria mostly depends on the application the RNG is intended for. So while for some uses in cryptography nothing less than an unpredictable RNG will do, for simulational purposes the main emphasis might be on high efficiency and repeatability.

Not all RNGs are designed to produce binary (bit) sequences. Besides bit generators, i.i.d. Uniform$\{0, \dots, 2^m - 1\}$ and i.i.d. Uniform$[0, 1)$ generators are also very common. The former produce m-bit-integers, where 31 and 32 bits are used quite frequently, the latter approximate reals, also only up to a certain precision (float, double, etc.), of course. These different outcomes can be transformed one into the other quite easily, though. Dividing a bit sequence into m-bit-blocks yields the binary representation of a sequence of m-bit-integers. An m-bit-integer divided by 2^m is a $[0, 1)$ real. The converse works just as well. In addition, a random binary sequence is purely random if and only if the corresponding m-bit-integer sequence is i.i.d. Uniform$\{0, \ldots, 2^m - 1\}$. Thus, even though some tests need bit sequences and others, say, 32-bit-integers as input, they can be applied to any RNG, as long as the output of the RNG is transformed to the right format.

Section 2 deals with statistical tests and special software packages designed for testing RNGs. Then, in section 3, the results of applying a particular test suite — TestU01 — to a family of RNGs currently being developed at the Centro Interdipartimentale Vito Volterra (CIVV) at the Università degli studi di Roma Tor Vergata, Italy, are presented.

2. Statistical Testing of RNGs

Bad RNGs are those that fail simple tests, whereas good RNGs fail only complicated tests that are hard to find and run.

P. L'Ecuyer

2.1. *Historical Development*

Over the years many statistical tests for testing random number generators have been proposed. One of the first collections was found in earlier editions of Knuth [1]. These tests, plus a few others designed for testing parallel generators, were implemented in *SPRNG: a scalable library for pseudorandom number generation* by Mascagni and Srinivasan [2].

New and more stringent[a] tests, compared to the ones from the just mentioned Knuth [1], were introduced by Marsaglia in 1985 [3]. Most of these tests were later implemented in *DIEHARD: A Battery of Tests of Randomness* by Marsaglia in 1995 [4], probably the best-known software package for RNG testing. Because of its very unflexible setup its usefulness has become rather limited, though, by now. First of all, the sample sizes (as well as other parameters) are fixed in the package and not very large by modern standards, which makes the test results unreliable for many of todays applications. Second the sequence must be provided to the package

[a]A test T_1 is considered more *stringent* than another test T_2, if a generator passing T_1 is also likely to pass T_2.

in the form of 32-bit-integers in a binary file, rather than just passing the RNG function to the package, which then in turn produces the numbers "on demand". This also makes any generator having an accuracy less than 32 bits fail DIEHARD.

The National Institute of Standards and Technology (NIST), USA, developed the *NIST Statistical Test Suite* by Rukhin et. al. [5] for the evaluation of the Advanced Encryption Standard (AES) candidate algorithms.

The state-of-the-art library for testing RNGs today is *TestU01: A C Library for Empirical Testing of Random Number Generators* introduced in 2007 by L'Ecuyer and Simard [6]. It implements: 1) a large variety of different RNGs proposed in the literature and/or used in software packages or operating systems, 2) most of the statistical tests from DIEHARD, the NIST package, the Knuth collection, other tests found in the literature and some original ones, 3) predefined test batteries and 4) tools for investigating dependence of the period length of a generator within a whole family of RNGs and the length of a sequence when this generator begins to fail a given test systematically.

Many of the statistical tests proposed for testing random numbers are easily passed while other tests seem to be quite difficult to pass. So what makes a good test? In the last years some efforts have been undertaken which, eventually, might lead to something like a "hierarchy of tests". For example, Tsang et. al. [7] define the so-called "Stringency level" of a test, in order to optimize the choice of parameters for the Collision Test. The results of testing a wide range of generators against *some difficult-to-pass tests of randomness* by Marsaglia and Tsang [8] as well as against other test suits, including DIEHARD, NIST and Knuth's collection are found in [9]. According to their results, applying these three difficult-to-pass tests seems to "reduce the volume and concentrate the essence" of many of the statistical tests in use today. However, through empirical investigations on our part, we have found generators, that do pass these three tests, but fail others quite badly. More details on these results are reported in section 3.

2.2. *General Setup of a Statistical Test*

What is the basic setup of a statistical test? Let $x_1, \ldots, x_N$ be a sequence of real numbers considered as realisations of random variables $X_1, \ldots, X_N$ over a common probability space $(\Omega, \mathcal{F}, P)$ where P is unknown. Denoting with $\mathcal{P}_0$ the set of all probability measures fulfilling certain given requirements about the common distribution of $X_1, \ldots, X_N$, the statement

$H_0 : P \in \mathcal{P}_0$ is called null hypothesis. Whether H_0 holds or not is unknown and the object of a test is to gather information from the sequence $x_1, \dots, x_N$ in order to either reject H_0 or not.

For the testing of binary sequences the null hypothesis throughout this paper will be given by $\mathcal{P}_0 = \{P : X_1, \dots, X_N$ is a truly random binary sequence$\}$, i.e.

$$H_0 : X_1, \dots, X_N \text{ i.i.d.},\ P(X_k = 1) = P(X_k = 0) = 1/2,\ \ k = 1, \dots, N. \tag{1}$$

Definition 1. The random number Y := t$(X_1, \dots, X_N)$, where t is a real-valued measurable map, is called statistic. A statistic T taking only the values 0 and 1 is called a test. Thereby, the events $\{T = 1\}$ and $\{T = 0\}$ are interpreted as "reject H_0" and "do not reject H_0", respectively. If the probability of rejecting H_0, even though it was true, is less than $\alpha \in (0, 1)$, i.e. $P_0(T = 1) \le \alpha$ for all $P_0 \in \mathcal{P}_0$, the test is called significance test at level α.

Remark 2. Let $T = t(X_1, \dots, X_N)$ be a significance test at level α.

(1) For a given realization $x_1, \dots, x_N$ H_0 is rejected, if $t(x_1, \dots, x_N) = 1$.
(2) Common choices for α are $0.05, 0.01$ or 0.001.
(3) Rejecting H_0 at level, say, $\alpha = 0.001$, does not mean that it is not true. It just means that it is very unlikely to be true, because if it was true, H_0 would only be rejected less than once in a thousand test runs, on average. It does happen nonetheless. On the other hand, not rejecting H_0 does not mean it is true. The sequence was just "not bad enough" to be rejected.

Now consider a statistic Y where the distribution under H_0 is known to have a distribution function $F_Y(y) := P_0(Y \le y)$. Then $U := F_Y(Y)$ is called p-value of the statistic Y. If F_Y is continuous, U has a Uniform$[0, 1]$ distribtion (under H_0). In this sense, p-values can be seen as standardized statistics. Moreover

$$P_0\big(\alpha/2 \le U \le 1 - \alpha/2\big) = \alpha \qquad \big(\alpha \in (0, 1), P_0 \in \mathcal{P}_0\big), \tag{2}$$

i.e.

$$T := \begin{cases} 0 & \alpha/2 \le U \le 1 - \alpha/2 \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

defines a significance test at level α. Thus, H_0 is rejected if the p-value U is close to 0 or 1. For general F_Y define a left and a right p-value of Y by

$$U_L := U = F_Y(Y) \qquad \text{and} \qquad U_R := F_Y^R(Y) \tag{4}$$

respectively, where $F_Y^R(y) := P_0(Y \geq y) = 1 - F_Y(y) + P_0(Y = y)$. Then reject H_0, if any of the left or right p-value is close to 0 (see L'Ecuyer and Simard [6]).

Example: The χ^2-goodness-of-fit test

Let $X_1, \dots, X_N$ be a sequence of i.i.d. copies of a random variable X and $H_0 : P = P_0$ the null hypothesis to be tested.

For $1 \leq k \leq K$ define $p_k := P_0(X \in A_k)$, where $A_1, \dots, A_K$ is a finite partition of $\mathbb{R}$ and let

$$C_k := \sum_{n=1}^{N} \mathbb{1}_{\{X_n \in A_k\}} \qquad (1 \leq k \leq K) \tag{5}$$

be the number of times the X_n fall into A_k, where $\mathbb{1}_B$ represents the indicator function of the event B. Then, under H_0, C_k follows a binomial distribution $B(N, p_k)$ and

$$Y := \sum_{k=1}^{K} \frac{(C_k - \mathbb{E}\, C_k)^2}{\mathbb{E}\, C_k} = \sum_{k=1}^{K} \frac{(C_k - Np_k)^2}{Np_k} \tag{6}$$

is approximately χ^2-distributed with $K-1$ degrees of freedom (as $N \to \infty$). Thus H_0 is rejected, if the p-value $U := F_{\chi^2(K-1)}(Y)$ is really close to 0 or 1, where $F_{\chi^2(r)}$ denotes the (continuous) distribution function of the χ^2-distribution with r degrees of freedom.

2.3. *Specialties for Testing of RNGs*

As seen in the previous subsection, any statistic Y, whose distribution under H_0 is known, at least approximately, can be used to define a test. Hence it will be called as a test statistic. In fact, for the testing of RNGs, a test can still be defined, even though the distribution of Y might not be (approximately) known. A number of generators may be used to estimate the distribution. And if these estimates are consistent across a variety of different and "presumably good" generators, the estimate may serve as H_0 target distribution.

In essence, statistical testing of RNGs is nothing but a particular kind of Monte Carlo simulation. Conversely, when testing an RNG for suitability with respect to a particular Monte Carlo problem, running the simulation with a related but simplified model, that is, one where the distribution of the result can be attained theoretically, may serve as a test. Even if the distribution is not known, the results of the designated RNG can still be compared to the ones produced by a few other "good" generators of quite different designs.

Most statistical tests for RNGs utilize the concept of a p-value. P-values of single tests should not only be in the proper range (not too close to 0 or 1), but should also be uniformly distributed on $[0, 1)$. Therefore, it might be useful to run the same test many times independently, i.e. on different parts of the original sequence. A number of independent first-level p-values can then be assessed by a second-level uniformity test resulting in an overall p-value. For example, while none of the single values of a sequence of supposedly independent p-values, say $0.24, 0.26, 0.23, 0.21$, gives any reason to question H_0, the whole sequence does reveal a very non-uniform and/or non-independent behaviour, though. This second-level test of uniformity could be the above-mentioned χ^2-goodness-of-fit test but usually an Anderson-Darling or a Kuiper version of the Kolmogorov-Smirnov test is applied.

Very often RNGs are tested against whole batteries of tests and therefore p-values close to 0 or 1 are not too uncommon even for good (including perfect) generators. Therefore, the choice of α could be somewhat different from the ones for usual statistical testing mentioned earlier. If the final (first- or second-level) p-value of a test is *really close* to 0 or 1, the RNG is said to fail the test. If the p-value is *suspicious*, the test is repeated and/or the sample size is increased and often things will then clarify. Otherwise the RNG is said to have passed the test. But remember that this does not mean that the null hypotheses is true. The meaning of *really close* and *suspicious* should be made clear before running the test, of course. Test batteries usually have some suggested values for that purpose.

For some applications there is the need for assessing much larger sequences than feasible due to memory limitations. These can be overcome by performing the same test many times on different subsequences and then somehow combining the results. One way is the above mentioned second-level uniformity test. But there is another useful approach. For most statistical tests the target (H_0-) distribution of the test statistic Y is either a normal or a χ^2- or a Poisson distribution. Thus, instead of calculating a p-value for each test and then performing a second-level test, all the single Y's could be added, the sum again being normal, χ^2 or Poisson, respectively. This sum statistic could then, in turn, be used to give the final p-value, see [10].

Example: The Birthday Spacings Test

This test was introduced by George Marsaglia in 1984 [3] and uses the fact that (for large n) the number of collisions among the spacings induced by the order statistics of m independent uniform $\{0, \ldots, n-1\}$ random variables is approximately Poisson with mean $\lambda = m^3/(4n)$. The proof of the corresponding limit theorem (Theorem 3 below) was never published, though. A different proof was provided by Klykova in 2002 [11].

Let $U_1, \ldots, U_m$ (the birthdays) be independent and uniformly distributed on $\{0, \ldots, n-1\}$ (the year),

$$0 =: U_{(0)} \leq U_{(1)} \leq \ldots \leq U_{(m)} \leq U_{(m+1)} := n-1 \tag{7}$$

the corresponding order statistics, $S_1, \ldots, S_{m+1}$ the induced spacings, i.e. $S_k := U_{(k)} - U_{(k-1)}$ and C the number of collisions among those spacings, i.e.

$$C := \sum_{k=1}^{m} \mathbb{1}_{\{S_{(k+1)} = S_{(k)}\}}, \tag{8}$$

where $\mathbb{1}_A$ stands for the indicator function of the event A and $S_{(1)}, \ldots, S_{(m+1)}$ are the order statistics of the spacings $S_1, \ldots, S_{m+1}$. The distribution of C we call birthday spacings collision distribution and we write $C \sim BSC(m, n)$.

Theorem 3. *If $C_n \sim BSC(m_n, n)$ for all $n \in \mathbb{N}$ such that $m_n^3/(4n) \underset{n\to\infty}{\longrightarrow} \lambda$ then*

$$P(C_n = k) \underset{n\to\infty}{\longrightarrow} \frac{\lambda^k}{k!} e^{-\lambda} \qquad (k \geq 0). \tag{9}$$

■

How does the test work? In [8] Marsaglia and Tsang suggest the following version of the Birthday Spacings Test. Choose $m = 2^{12} = 4096$, $n = 2^{32}$ and therefore $\lambda = 4$. Let the RNG generate 4096 32-bit-integers, sort them and take differences to get the spacings. Then sort the spacings and count how many times adjacent spacings are equal. This gives the number of collisions. Repeat this process 5000 times to get a realization of a sequence of supposedly independent Poisson random variables with mean $\lambda = 4$. Then perform a χ^2-goodness-of-fit-test as follows: For $k = 0, \ldots, 9$

let N_k be the number of times there were k collisions among the 5000 runs and let N_{10} be the number of times there were more than 9 collisions. Also let $E_k := 5000p_k$ be the expected number of N_k (under H_0), where $p_k := \frac{\lambda^k}{k!} e^{-\lambda}$ $(k = 0, \ldots, 9)$ is the probability of a Poisson random variable taking the value k, $p_{10} := 1 - \sum_{k=0}^{9} p_k$ respectively. Then

$$Y := \sum_{k=0}^{10} \frac{(N_k - E_k)^2}{E_k} \tag{10}$$

follows approximately a χ^2-distribution with 10 degrees of freedom. Therefore the test is failed if the p-value $U := F_{\chi^2(10)}(Y)$ is really close to 0 or 1, where $F_{\chi^2(r)}$ denotes the (continuous) distribution function of the χ^2-distribution with r degrees of freedom.

Instead of performing a χ^2-goodness-of-fit test on those 5000 collision counts, they could also be added, the sum being Poisson with mean $4 * 5000 = 20000$. Even though it might not be feasible to calculate the corresponding p-value directly, this could be done by a normal approximation.

In [10], L'Ecuyer and Simard design birthday spacings tests for testing uniformity in d dimensions by dividing the corresponding d-dimensional hypercube into hyperboxes of equal size and then enumerating them in the natural order. This kind of tests is implemented in TestU01 [6].

3. Test Results

Good statistical properties of the sequences it produces being not sufficient, but certainly necessary, any RNG aspiring to be called good has to undergo some intensive and extensive empirical testing. In this section we present the results of applying the test batteries from the RNG software package TestU01 to a family of generators first introduced by Abundo, Accardi and Auricchi [12] and currently further being developed at the Centro Interdipartimentale Vito Volterra (CIVV) in Roma, Italy. They are also compared to the results of a few other popular generators, namely, the Wichmann-Hill-generator [13] used in Microsoft Excel 2003, CombLec88 - a combination of two linear congruential generators used in RANLIB, CERNLIB, Boost, Octave and Scilab proposed by L'Ecuyer [14], KISS99 - a combination of four different generators designed by Marsaglia [15], the 2002 version of the Mersenne Twister generator and ISAAC, proposed by Jenkins for use in cryptography [16]. The MT19937_2002 generator is described in [17], has good statistical properties and has become quite popular in software for numerical simulation (like MATLAB, for example). It is not unpredictable, though, and thus not recommended for cryptographical purposes.

Table 1. Results of the **test battery Crush** (software package TestU01 1.2.1). It produces **144 p-values from 96 independent tests**, thereby sampling around **128 GByte** of random bits (> **15 double layer DVDs**).

Generator	Clear failures	Suspect (of which failed)	Total failed
QPdyn	0	3 (0)	0
QPdyn-s	6	12(8)	14
MSExcel_2003	12	7 (*)	12*
CombLec88	1	5 (1)	2
Kiss99	0	2 (0)	0
MT19937_2002	2	4 (0)	2
ISAAC	0	3 (0)	0

Note: * no follow-up testing done

Table 1 shows the results of applying the test battery Crush from the RNG software package TestU01 to QPdyn and QPdyn-s. Each test from the battery results in (at least) one p-value, a number in $[0, 1]$. It is interpreted according to table 2.

Table 2. Interpretion of p-values.

p-value	Interpretation
$0.01 < p < 0.99$:	Clear passed
p or $(1 - p) < 10^{-10}$:	Clear failure
Otherwise:	Suspect behaviour, repeat test 3 times independently, if test continues to show suspect results, it is considered as failed, otherwise passed

As mentioned in section 2, Marsaglia and Tsang [8, 9] proposed three tests, that they consider to be a concentration of DIEHARD and other statistical tests used today. However, the generator QPdyn-s passes these three tests, but at the same time fails the test battery Crush quite badly, as can be seen from table 1. So this might be a starting point for comparing this Marsaglia-and-Tsang-three-tests-battery with the battery Crush for many different generators, to decide which is more stringent. Or, more likely, they will be found to be somewhat complementary in the sense that both batteries are specialized in finding certain deficiencies, which the other one does not detect. In this case an appropriate combination of both batteries should be considered.

4. Summary and Outlook

The software package TestU01 should be considered THE STANDARD for testing RNGs. The particular version of QP-Dyn tested in the previous section passes all of the tests. But this is only the beginning. Eventually, QP-Dyn is intended to become an algorithm that automatically but somewhat randomly chooses parameters to generate not random numbers but random number generators. The idea of using computers to find good parameters for families of generators is also mentioned in section 3.7 of [18].

In a forthcoming paper we will study how to condense statistical testing in order to test QP-Dyn considered as a random RNG generator.

Acknowledgements

This work was supported by the Centro Interdipartimentale Vito Volterra at the Università degli studi di Roma Tor Vergata, Italy and the European Union Research Training Network "Quantum Probability with Applications to Physics, Information Theory and Biology".

References

1. Donald E. Knuth. *The art of computer programming. Vol. 2: Seminumerical algorithms. 3rd ed.* Bonn: Addison-Wesley. xiii, 762 p. , 1998.
2. Michael Mascagni and Ashok Srinivasan. Algorithm 806: SPRNG: a scalable library for pseudorandom number generation. *ACM Transactions on Mathematical Software*, 26(3):436–461, September 2000. See correction [19].
3. George Marsaglia. A current view of random number generators. In L. Billard, editor, *Computational Science and Statistics: The Interface*, pages 3–10, Amsterdam, 1985. Elsevier Press.
4. George Marsaglia. DIEHARD: A battery of tests of randomness, 1996.
5. A. Rukhin et. al. *A statistical test suite for random and pseudorandom number generators for cryptographic applications.* NIST Special Publication 800-22, National Institute of Standards and Technology (NIST), Gaithersburg, Maryland, USA, 2001.
6. Pierre L'Ecuyer and Richard Simard. TestU01: A C library for empirical testing of random number generators. *ACM Transactions on Mathematical Software*, 2007.
7. W. W. Tsang, L. C. K. Hui, K. P. Chow, C. F. Chong, and C. W. Tso. Tuning the collision test for power. In *ACSC '04: Proceedings of the 27th Australasian conference on Computer science*, pages 23–30, Darlinghurst, Australia, Australia, 2004. Australian Computer Society, Inc.
8. George Marsaglia and Wai Wan Tsang. Some difficult-to-pass tests of randomness. *Journal of Statistical Software*, 7(3):1–8, January 2002.
9. Wai Wan Tsang. Development of cryptographic random number generators.

Technical report, The University of Hong Kong, Department of Computer Science and Information Systems, August 2003.

10. Pierre L'Ecuyer and Richard Simard. On the performance of birthday spacings tests with certain families of random number generators. *Math. Comput. Simul.*, 55(1-3):131–137, 2001.
11. N. V. Klykova. Limit distribution of a number of coinciding intervals. *Theory Probab. Appl.*, 47(1):151–156, 2002.
12. M. Abundo, Luigi Accardi, and A. Auricchio. Hyperbolic automorphisms of tori and pseudo-random sequences. *Calcolo*, 29(3-4):213–240, 1992.
13. B. A. Wichmann and I. D. Hill. An efficient and portable pseudo-random number generator. *Applied Statistics*, 31:188–190, 1982. See also corrections and remarks in the same journal by Wichmann and Hill, **33** (1984) 123; McLeod **34** (1985) 198–200; Zeisel **35** (1986) 89.
14. P. L'Ecuyer. Efficient and portable combined random number generators. *Communications of the ACM*, 31(6):742–749 and 774, 1988. See also the correspondence in the same journal, 32, 8 (1989) 1019–1024.
15. G. Marsaglia. Random numbers for C: The END? Posted to the electronic billboard `sci.crypt.random-numbers`, January 20 1999.
16. B. Jenkins. ISAAC. In Dieter Gollmann, editor, *Fast Software Encryption, Proceedings of the Third International Workshop, Cambridge, UK*, volume 1039 of *Lecture Notes in Computer Science*, pages 41–49. Springer-Verlag, 1996. `http://burtleburtle.net/bob/rand/isaacafa.html`.
17. Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, 1998.
18. P. L'Ecuyer. Uniform random number generation. In S. G. Henderson and B. L. Nelson, editors, *Simulation*, Handbooks in Operations Research and Management Science, chapter Chapter 3, pages 55–81. Elsevier, Amsterdam, The Netherlands, 2006.
19. Michael Mascagni and Ashok Srinivasan. Corrigendum: Algorithm 806: SPRNG: a scalable library for pseudorandom number generation. *ACM Transactions on Mathematical Software*, 26(4):618–619, December 2000. See [2].