On testing pseudorandom generators via statistical tests based on the arcsine law[☆]

Paweł Lorek[a], Grzegorz Łoś[b], Karol Gotfryd[c], Filip Zagórski[c]

[a]*Mathematical Institute, University of Wrocław, pl. Grunwaldzki 2/4, 50-384, Wrocław, Poland*
[b]*Institute of Computer Science, University of Wrocław, Joliot-Curie 15, 50-383, Wrocław, Poland*
[c]*Department of Computer Science, Faculty of Fundamental Problems of Technology, Wrocław University of Science and Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland*

Abstract

Testing the quality of pseudorandom number generators is an important issue. Security requirements become more and more demanding, weaknesses in this matter are simply not acceptable. There is a need for an in-depth analysis of statistical tests – one has to be sure that rejecting/accepting a generator as good is not a result of errors in computations or approximations. In this paper we propose a second level statistical test based on the arcsine law for random walks. We provide a Berry-Esseen type inequality for approximating the arcsine distribution, what allows us to perform a detailed error analysis of the proposed test.

Keywords: The arcsine law, Random walks, Pseudorandom number generator, Statistical testing, Second level testing, Berry-Esseen type inequality, Randomness, Dyck paths

1. Introduction

Random numbers are key ingredients in various applications, *e.g.,* in cryptography (*e.g.,* for generating cryptographic keys) or in simulations (*e.g.,* in Monte Carlo methods), just to mention a few. No algorithm can produce truly random numbers. Instead, pseudorandom number generators (PRNGs) are used. These are deterministic algorithms producing numbers which we expect to resemble truly random ones *in some sense*. There are two classes of tests used to evaluate PRNGs, theoretical and statistical ones. Theoretical tests examine the intrinsic structure of a given generator, the sequence does not necessarily need to be generated. Two classical examples are the *lattice test* [1] and the *spectral test* described in [2] (Section 3.3.4). See also [3] for a description of some standard tests from this class. This category of tests is very specific to each family of generators *e.g.,* some are designed only for linear congruential generators. On the other hand, the second class of tests – empirical tests – are conducted on a sequence generated by a PRNG and require no knowledge of how it was produced. The main goal of these tests is to check if the sequence of numbers $\mathbf{U} = (U_1, U_2, \ldots, U_n)$ (or bits, depending on the actual implementation) produced by a PRNG has properties similar to those of a sequence generated truly at random. These tests try to find statistical evidence against the null hypothesis $\mathcal{H}_0$ stating that the sequence is a sample from independent random variables with uniform distribution. Any function of a finite number of uniformly distributed random variables, whose (sometimes approximate) distribution under hypothesis $\mathcal{H}_0$ is known, can be used as a statistical test. Due to the popularity and significance of the problem, a variety of testing procedures have been developed in recent years. Such statistical tests aim at detecting various deviations in generated sequences, what allows for revealing flawed PRNGs producing predictable output. Some of the procedures encompass classical tools from statistics like the Kolmogorov-Smirnov test or the Pearson's chi-squared test,

[☆]Work supported by NCN Research Grant DEC-2013/10/E/ST1/00359
Email addresses: Pawel.Lorek@math.uni.wroc.pl (Paweł Lorek), grzegorz314@gmail.com (Grzegorz Łoś), Karol.Gotfryd@pwr.edu.pl (Karol Gotfryd), Filip.Zagorski@pwr.edu.pl (Filip Zagórski)

which are used for comparing the theoretical and empirical distributions of appropriate statistics calculated for a PRNG's output. It is also possible to adapt tests of normality like the Anderson-Darling or Shapiro-Wilk tests for appropriately transformed pseudorandom sequences. These methods exploit the properties of sequences of i.i.d. random variables. Based on the original sequence $\mathbf{U}$ returned by the examined PRNG we are able to obtain realizations of random variables with known theoretical distributions. Some examples of probabilistic laws used in practice in this kind of tests can be found *e.g.,* in [2]. They include such procedures like the gap test, the permutation test and the coupon collector's test, just to name a few (see [2] for a more detailed treatment). These methods have also the advantage that they implicitly test the independence of the generator's output. The main issue with such methods is that a single statistical test looks only at some specific property that holds for sequences of truly random numbers. Hence, for practical purposes bundles of diverse tests are created. Such a test bundle consists of a series of individual procedures based on various stochastic laws from probability theory. A PRNG is then considered as good if the pseudorandom sequences it produces pass all tests in a given bundle. Note that formally it proves nothing, but it increases the confidence in the simulation results. Thus, they are actually tests for *non-randomness*, as pointed out in [4]. Some examples of such test suites are Marsaglia's Diehard Battery of Tests of Randomness from 1995, Dieharder developed by Brown et al. (see [5]), `TestU01` implemented by L'Ecuyer and Simard (see [6, 7]) and NIST Test Suite [8]. The last one, designed by the National Institute of Standard and Technology, is currently considered as one of the state of the art test bundles. It is often used for the preparation of many formal certifications or approvals.

A result of a single statistical test is typically given in the form of a p-value, which, informally speaking, represents the probability that a perfect PRNG would produce "less random" sequence than the sequence being tested w.r.t. the used statistic. We then reject $\mathcal{H}_0$ if $p < \alpha$, where α is the *significance level* (usually $\alpha = 0.01$) and accept $\mathcal{H}_0$ if $p \geq \alpha$. Such an approach is usually called *one level* or *first level* test. Although the interpretation of a single p-value has a clear statistical explanation, it is not quite obvious how to interpret the results of a test bundle, *i.e.,* of multiple tests. Under $\mathcal{H}_0$ the distribution of p-values is uniform. However, in a test bundle several different tests are applied to the same output of a PRNG, hence the results are usually correlated. The documentation of the NIST Test Suite includes some clues on how to interpret the results of their bundle (Section 4.2 in [9]), but in the introduction it is frankly stated: "It is up to the tester to determine the correct interpretation of the test results".

To disclose flaws of PRNGs, a very long sequence is often required. In such situations, the applicability of a statistical test can be limited (depending on the test statistic) by the memory size of the computer. An alternative approach is to use a so-called *two level* (a term used *e.g.,* in [3]) or *second level* (a term used *e.g.,* in [4, 10]) test. In this approach we take into account several results from the same test over disjoint sequences generated by a PRNG. We obtain several p-values which are uniformly distributed under $\mathcal{H}_0$, what is tested by *e.g.,* some goodness-of-fit test (with potentially different level of significance – NIST suggests to use 0.0001 – obtaining new "final" p-value). The authors in [11] observed that this method may be comparable to a first level test in terms of the power of a test (informally speaking, it represents the probability of observing "less random" sequence than the sequence being tested under an *alternative* hypothesis $\mathcal{H}_1$, see [11] for details), but often it produces much more *accurate* results, as shown in [4]. Roughly speaking, the accuracy is related to the ability, given a non-random PRNG, of recognizing its sequences as non-random (for details see [4]). We will follow this approach.

In the second level approach one has to take under consideration the approximation errors in the computation of a p-value. For example, in a first level test one usually calculates a p-value of a statistic which – under $\mathcal{H}_0$ – is *approximately* normally distributed. The approximation comes then from the central limit theorem, which lets us substitute the distribution of a given sum with the standard normal distribution. These errors in calculations of individual p-values may accumulate, resulting in an error of a p-value in a second level test, thus making the test *not reliable*. Following [4] we say that the second level test is *not reliable* when, due to errors or approximations in the computation of p-values (in the first level), the distribution of p-values is not uniform under $\mathcal{H}_0$. Fortunately, this approximation error can be bounded using the Berry-Esseen inequality and the final error of a second level test can be controlled (see [4, 10] for a detailed example based on the binary matrix rank test). The influence of approximations on the computation of p-values in a second level test was also considered in [12, 13]. In this article we present a statistical test based

on the arcsine law, in which at some point we approximate a distribution of some random variable with the arcsine distribution. We provide a Berry-Esseen type inequality which upper bounds the approximation error, what allows us to control the reliability of our second level test.

An interesting approach for testing PRNGs was presented by Kim et al. in [14]. The concept of their tests is based on the properties of a random walk (the gambler's ruin algorithm) on the cyclic group $\mathbb{Z}_n = \{0, \ldots, n-1\}$ with 0 being an absorbing state – more precisely, on the time till absorption. The authors in [14] propose three different variants of the test. The general idea of the basic procedure is the following. For some fixed $p \in (0,1)$ and $x \in \mathbb{Z}_n$, the output $\mathbf{U} = (U_i)$ of a PRNG is treated as numbers from the unit interval and used to define a random walk starting in x such that if $U_i < p$ and the process is in state s, then it moves to $(s+1) \mod n$, otherwise it moves to $(s-1) \mod n$. The aim of this test is to compare the theoretical and the empirical distributions of the time to absorption in 0 when starting at x. Based on the values of testing statistic, the PRNG is then either accepted or rejected. The authors reported some "hidden defects" in the widely used Mersenne Twister generator. However, one has to be very careful when dealing with randomness. It seems like re-seeding a PRNG with a fixed seed is an error which can lead to wrong conclusions. The criticism was raised by Ekkehard and Grønvik in [15], where the authors also showed that the *properly* performed tests of Kim et al. [14] do not reveal any defects in the Mersenne Twister PRNG. Recently, the authors in [16] have proposed another gambler's ruin based procedure for testing PRNGs. In their method they exploited formulas for winning probabilities for arbitrary sequences $p(i)$ and $q(i)$, (*i.e.,* the winning and losing probabilities depend on the current fortune) which are the parameters of the algorithm.

In recent years a novel kind of testing techniques has been introduced for more careful verification of generators. The core idea of this class of methods is based on an observation that the binary sequence (B_i) produced by a PRNG, after being properly rescaled, can be interpreted as an one-dimensional random walk $(S_n)_{n\in\mathbb{N}}$ with $S_k = \sum_{i=1}^{k} X_i$, where $X_i = 2B_i - 1$. For random walks defined by truly random binary sequences a wide range of statistics have been considered over the years and a variety of corresponding stochastic laws have been derived (see *e.g.,* [17]). For a good PRNG we may expect that its output will behave like S_n. Hence, the following idea comes to mind: choose some probabilistic law that holds for truly random bit sequences and compare the theoretical distribution of the corresponding statistic with the empirical distribution calculated for m sequences produced by a given PRNG in m independent experiments. This comparison can be done *e.g.,* by computing the p-value of an appropriate test statistic under the null hypothesis that the sequence generated by this PRNG is truly random.

Another concept named *statistical distance based testing* was suggested in [18]. It relies on calculation of some statistical distances like *e.g.,* total variation distance between the theoretical and empirical distributions for considered characteristics and rejecting a PRNG if the distance exceeds some threshold. We will also follow this approach, indicating the corresponding threshold. In [18] the authors derive their test statistics from the law of iterated logarithm for random walks (the procedure is called the LIL test). The proposed by us procedure uses similar methodology and is based on the arcsine law. We made the code publicly available, see [19]. It includes the arcsine law based as well as the law of iterated logarithm based statistical tests, the implementation of many PRNGs (more than described in this article) including the Flawed generator (see Section 4) and the seeds we used.

Organization of the paper. In the following Section 2 we define a general notion of a PRNG and recall the aforementioned stochastic laws for random walks. The testing method along with the error analysis is described in Section 3. The concise report on experimental results (including the Flawed generator introduced in Section 4) is given in Section 5. In Section 6 we mention other implementations of the tests based on the arcsine law. We conclude in Section 7.

2. Pseudorandom generators and stochastic laws for random walks

2.1. Pseudorandom generators

The intuition behind *pseudorandom number generator* is clear. However, let us give a strict definition roughly following Asmussen and Glynn [20].

Definition 2.1. *A Pseudorandom number generator (PRNG) is a 5-tuple $< E, V, s_0, f, g >$, where E is a finite state space, V is a set of values, $s_0 \in E$ is a so-called seed, i.e., an initial state in the sequence $(s_i)_{i=0}^{\infty}$, a function $f : E \to E$ describes the transition between consecutive states $s_n = f(s_{n-1})$ and $g : E \to V$ maps the generator's state into the output.*

Usually $V = (0, 1)$ or $V = \{0, 1, \ldots, M-1\}$ for some $M \in \mathbb{N}$, the latter one is used throughout the paper. Recall that LCG (linear congruential generator) is a generator which updates its state according to the formula $s_n = (as_{n-1} + c) \bmod M$. Thus, it is defined by three integers: a modulus M, a multiplier a, and an additive constant c. In the case $c = 0$, the generator is called MCG (multiplicative congruential generator). For a detailed description of some commonly used PRNGs see the surveys [21, 22, 23] or the book [24].

It is clear that both the input and the output of a random number generator can be viewed as a finite sequence of bits. For a PRNG to be considered as good, the output sequences should have some particular property, namely each returned bit has to be generated independently with equal probability of being 0 and 1. We say that the sequence of bits is *truly random* if it is a realization of a Bernoulli process with success probability $p = \frac{1}{2}$.

Given a PRNG G returning integers from the set V, we may obtain a pseudorandom binary sequence with any given length using the following simple procedure. Namely, as long as the bit sequence s is not sufficiently long, generate the next pseudorandom number a and append its binary representation (on $\lceil \log_2 M \rceil$ bits) to the current content of s. In the ideal model with G being truly random number generator, such algorithm produces truly random bit sequences provided that M is a power of 2. Indeed, for $M = 2^k$ there is one to one correspondence between k-bit sequences and the set V. Hence, if each number is generated independently with uniform distribution on V, then each combination of k bits is equally likely and therefore each bit of the output sequence is independent and equal to 0 or 1 with probability $\frac{1}{2}$.

However, this is not true for $M \neq 2^k$. It is easy to observe that in such a case the generator is more likely to output 0s and the generated bits are no longer independent. Thus, rather than simply outputting the bits of a, one may instead take d first bits from the binary representation of $\frac{a}{M}$ for some fixed d. Such a method has the advantage that it can be easily adopted for an underlying generator returning numbers from the unit interval, what is common for many PRNG implementations.

2.2. Stochastic laws for random walks

Let $(B_i)_{i \geq 0}$ be a *Bernoulli process* with a parameter $p \in (0, 1)$, *i.e.,* a sequence of independent random variables with identical distribution $P(B_1 = 1) = 1 - P(B_1 = 0) = p$. A good PRNG should *behave* like a generator of Bernoulli process with $p = 1/2$ (what we assume from now on). It will be, however, more convenient to consider the following transformed process

$$X_i = 2B_i - 1, \quad S_0 = 0, \quad S_k = \sum_{i=1}^{k} X_i, \ k = 1, \ldots \tag{1}$$

The sequence X_i is $\{-1, +1\}$-valued, the process $(S_n)_{n \in \mathbb{N}}$ is called a random walk.

The law of iterated logarithm. Of course $|S_n| \leq n$. However, large values of $|S_n|$ occur with small probability and the values of S_n are in practice in a much narrower range than $[-n, n]$. *The weak* and *the strong law of large numbers* imply that $\frac{S_n}{n} \xrightarrow{P} 0$, and even $\frac{S_n}{n} \xrightarrow{a.s.} 0$, where $\xrightarrow{P}$ denotes the *convergence in probability* and $\xrightarrow{a.s.}$ denotes the *almost sure convergence*. Thus, the deviations of S_n from 0 grow much slower than linearly. On the other hand *the central limit theorem* states that $\frac{S_n}{n} \xrightarrow{D} \mathcal{N}(0, 1)$ (where $\xrightarrow{D}$ denotes the convergence in distribution), what is in some sense a lower bound on fluctuations of S_n – they will leave the interval $[-\sqrt{n}, \sqrt{n}]$ since we have $\limsup_{n \to \infty} \frac{S_n}{\sqrt{n}} = \infty$ (implied by 0-1 Kolmogorov's Law, see *e.g.,* Theorem 5.1 in [25]). It turns out that the fluctuations can be estimated more exactly.

Theorem 2.2 (The law of iterated logarithm, [26], cf. also Chapter VIII.5 in [17])**.** *For a random walk S_n we have*

$$\mathbb{P}\left(\liminf_{n\to\infty} \frac{S_n}{\sqrt{2n\log\log n}} = -1\right) = 1,$$
$$\mathbb{P}\left(\limsup_{n\to\infty} \frac{S_n}{\sqrt{2n\log\log n}} = +1\right) = 1.$$

Thus, to normalize S_n dividing by n is *too strong* and dividing by $\sqrt{n}$ is *too weak*. The fluctuations of S_n from 0 grow proportionally to $\sqrt{2n\log\log n}$.

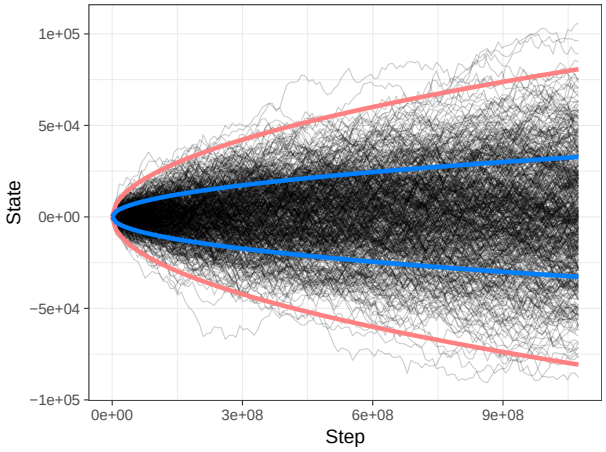

Figure 1: 500 trajectories of random walks of length 2^{30}. Blue plot: $\pm\sqrt{n}$, red plot: $\pm\sqrt{2n\log\log n}$

To depict the law of iterated logarithm, we took 500 output sequences $\mathcal{B}^1, \ldots, \mathcal{B}^{500}$ from the Mersenne Twister MT19937 generator, each initialized with a random seed taken from `http://www.random.org`, where each output $\mathcal{B}^j = (B_1^j, \ldots, B_n^j), B_i^j \in \{0,1\}, j = 1, \ldots, 500, i = 1, \ldots, n$ was of length $n = 2^{30}$. In Figure 1 we presented these 500 trajectories $(k, S_k^j), j = 1, \ldots, 500, k = 0, \ldots, n$, where $S_k^j = \sum_{i=1}^{k}(2B_i^j - 1)$. Each trajectory is depicted by a single polyline. The darker the image the higher the density of trajectories. We can see that $\pm\sqrt{2n\log\log n}$ roughly corresponds to the fluctuations of S_n. However, few trajectories after around billion steps are still outside $[-\sqrt{2n\log\log n}, \sqrt{2n\log\log n}]$. The law of iterated logarithm tells us that for *appropriately* large n the trajectories will not leave $[-\sqrt{2n\log\log n}, \sqrt{2n\log\log n}]$ with probability 1, what is not the case in Figure 1. It means that n must be much larger than 2^{30}.

One could think that the following is a good test for randomness: fix some number, say 100, and classify the considered PRNG as good if the difference between the number of ones and zeros never exceeds 100. The large difference may suggest that zeros and ones have different probabilities of occurrence. However, the law of iterated logarithm tells us that this reasoning is wrong. Indeed, we *should* expect some fluctuations and the absence of them means that a PRNG does not produce bits which can be considered random. This property of random walks was used by the authors in [18] for designing a novel method of testing random number generators.

There is yet another interesting property. Define $S_n^{lil} = \frac{S_n}{\sqrt{2n\log\log n}}$. The law of iterated logarithm implies that S_n^{lil} does not converge pointwise to any constant. However, it converges to 0 in probability. Let us fix some small $\varepsilon > 0$. For almost all n, with an arbitrary high probability $p < 1$ the process S_n^{lil} will not leave $(-\varepsilon, \varepsilon)$. On the other hand, this tells us that the process will be outside this interval infinitely many times. This apparent contradiction shows how can our intuition be unreliable on phenomena taking place at infinity.

The arcsine law. The observations described previously imply that *averaging every* S_n, it will spend half of its time above the x-axis and half of its time below. However, the typical situation is counter-intuitive (at

first glance): typically the random walk will either spend most of its time above or most of its time below the x-axis. This is expressed in the Theorem 2.3 below (for reference see *e.g.,* [17]). Before we formulate the theorem, let us first introduce some notations. For a sequence $X_1, X_2, \ldots$, as defined in (1), let

$$D_k = \mathbb{1}\left(S_k > 0 \vee S_{k-1} > 0\right), k = 1, 2, \ldots, \tag{2}$$

where $\mathbb{1}(\cdot)$ is the indicator function. D_k is equal to 1 if the number of ones exceeds the number of zeros either at step k or at step $k-1$, and 0 otherwise (in a case of ties, *i.e.,* $S_k = 0$, we look at the previous step letting $D_k = D_{k-1}$). In other words, $D_k = 1$ corresponds to the situation in which the line segment of the trajectory of the random walk between steps $k-1$ and k is above the x-axis.

Theorem 2.3 (The arcsine law)**.** *Let $(B_i)_{i\geq 0}$ be a Bernoulli process. Define $X_i = 2B_i - 1$ and $L_n = \sum_{k=1}^{n} D_k$ (D_k is given in (2)). For $x \in (0,1)$ we have*

$$\mathbb{P}\left(L_n \leq x \cdot n\right) \xrightarrow[n\to\infty]{} \frac{1}{\pi}\int_0^x \frac{dt}{\sqrt{t(1-t)}} = \frac{2}{\pi}\arcsin\sqrt{x}\ .$$

The probability $\mathbb{P}\left(L_n \leq x \cdot n\right)$ is the chance that the random walk was above the x-axis for at most x fraction of the time. The limiting distribution is called the *arcsine distribution*. Its density function is given by $f^{asin}(t) = \frac{1}{\pi}\sqrt{t(1-t)}$ and the cumulative distribution function (cdf) is $F^{asin}(t) = \frac{2}{\pi}\arcsin\sqrt{t}$. The shape of the pdf $f^{asin}(t)$ clearly indicates that the fractions of time spent above and below the x-axis are more likely to be unequal than close to each other.

3. Testing PRNGs based on the arcsine law

In this Section we will show how to exploit the theoretical properties of random walks from the preceding discussion to design a practical routine for testing PRNGs. We describe our approach based on the arcsine law which we employ for experimental evaluation of several commonly used generators (the results are presented in Section 5). We also perform an error analysis of the proposed testing procedure, providing corresponding bounds on the approximation errors. Finally, we make some remarks on the reliability of our second level test.

3.1. The arcsine law based testing

The general idea of tests is the following. Take a sequence of bits generated by PRNG, rescale them as in (1) and compare the empirical distribution of

$$S_n^{asin} = \frac{1}{n}\sum_{k=1}^{n} D_k \in [0,1]$$

(a fraction of time instants at which ones prevail zeros) with its theoretical distribution assuming that truly random numbers were generated. In terms of hypothesis testing: given the null hypothesis $\mathcal{H}_0$ that the bits in the sequence were generated independently and uniformly at random (vs. $\mathcal{H}_A$: that the sequence was not randomly generated), the distribution of S_n^{asin} follows the arcsine law (Theorem 2.3), *i.e.,* we can conclude that *for large* n we have

$$\mathbb{P}\left(S_n^{asin} \leq x | \mathcal{H}_0\right) \approx \frac{1}{\pi}\int_0^x \frac{dt}{\sqrt{t(1-t)}} = \frac{2}{\pi}\arcsin(\sqrt{x}) \tag{3}$$

(we will be more specific on "$\approx$" in Section 3.2). We follow the *second level* testing approach (cf. [4, 10]), *i.e.,* we take into account several results from the same test over different sequences. To test a PRNG we generate m sequences of length n each, thus obtaining m realizations of the variable S_n^{asin}. Denoting by

$S_{n,j}^{asin}$ the value of j-th simulation's result (we call them a *basic tests*), we then calculate the corresponding p-values

$$p_j = \mathbb{P}\left(S_n^{asin} > S_{n,j}^{asin} | \mathcal{H}_0\right) = 1 - \frac{2}{\pi} \arcsin\left(\sqrt{S_{n,j}^{asin}}\right), j = 1, \ldots, m$$

Under $\mathcal{H}_0$ the distribution of $p_j, j = 1, \ldots, m$, should be uniform on $[0, 1]$. We fix some partition of $[0, 1]$ and count the number of p-values within each interval. In our tests we will use an $(s+1)$-element partition $\mathcal{P}_s = \{P_1, \ldots, P_{s+1}\}$, where

$$\begin{aligned} P_1 &= \left[0, \frac{1}{2s}\right), \\ P_i &= \left[\frac{2i-3}{2s}, \frac{2i-1}{2s}\right), \quad 2 \le i \le s, \\ P_{s+1} &= \left[1 - \frac{1}{2s}, 1\right]. \end{aligned}$$

Now we define the measures μ_m (the uniform measure on $\mathcal{P}_s$), ν_n (the empirical measure on $\mathcal{P}_s$), E_i (the expected number of p-values within P_i) and O_i (the number of observed p-values within P_i). For $1 \le i \le s+1$ let

$$\mu_m\left(P_i\right) = \begin{cases} \frac{1}{s} & \text{if } i \in \{2, \ldots, s\} \\ \\ \frac{1}{2s} & \text{if } i \in \{1, s+1\} \end{cases}, \quad E_i = \begin{cases} \frac{m}{s} & \text{if } i \in \{2, \ldots, s\} \\ \\ \frac{m}{2s} & \text{if } i \in \{1, s+1\} \end{cases}$$

$$\nu_m\left(P_i\right) = \frac{|\{j : \ p_j \in P_i, 1 \le j \le m\}|}{m}, \quad O_i = m \cdot \nu_m(P_i).$$

We perform the Pearson's goodness-of-fit test, which uses the following test statistic

$$T^{asin} = \sum_{i=1}^{s+1} \frac{(O_i - E_i)^2}{E_i} = m \cdot \sum_{\mathcal{A} \in \mathcal{P}_s} \frac{(\mu_m(\mathcal{A}) - \nu_m(\mathcal{A}))^2}{\mu_m(\mathcal{A})}.$$

Under the null hypothesis, T^{asin} has approximately the chi-squared distribution with s degrees of freedom. We calculate the corresponding p-value

$$p_{\chi^2} = \mathbb{P}\left(X > T^{asin}\right),$$

where X has a $\chi^2(s)$ distribution. Large values of T^{asin} – and thus small values of p_{χ^2} – let us suspect that a given PRNG is not good. Typically, we reject $\mathcal{H}_0$ (*i.e.,* we consider the test *failed*) if $p_{\chi^s} < \alpha$, where α is a predefined *level of significance* (for a second level test we use $\alpha = 0.0001$, as suggested by NIST). Note that the probability of rejecting $\mathcal{H}_0$ when the sequence is generated by a perfect random generator (so-called *Type I error*) is exactly α.

Another approach relies on the *statistical distance based testing*, which is the technique presented in [18]. We consider the statistic

$$d_{tv}^{asin} = \frac{1}{2} \sum_{\mathcal{A} \in \mathcal{P}_s} |\mu_m(\mathcal{A}) - \nu_m(\mathcal{A})| \in [0, 1],$$

i.e., a total variation distance between the theoretical distribution μ_m and the empirical distribution ν_m. Similarly, large values of d_{tv}^{asin} indicate that a given PRNG is not good. Concerning *Type I error* we will make use of the following lemma (see Lemma 3 in [27] or its reformulation, Lemma 1 in [28]).

Lemma 3.1. *Assume $\mathcal{H}_0$ and consider the partition $\mathcal{P}_s$. Then, for all $\varepsilon \ge \sqrt{20(s+1)/m}$ we have*

$$\mathbb{P}\left(2d_{tv}^{asin} > \varepsilon | \mathcal{H}_0\right) \le 3 \exp\left(-\frac{m\varepsilon^2}{25}\right).$$

To summarize, for a given PRNG we generate m sequences of length n each. and we choose s (and thus the partition $\mathcal{P}_s$). We then calculate d_{tv}^{asin} and T^{asin} together with its p_{χ^2}-value. We specify the thresholds for p_{χ^2}-value and d_{tv}^{asin} indicating whether the test *failed* or not (the details are presented in Section 5). We denote the described procedure as the ASIN test.

Remark. Note that the described procedure for calculating T^{asin} and d_{tv}^{asin} is equivalent to the following one. Instead of calculating p-values of $S_{n,j}^{asin}$, we could directly count the number of $S_{n,j}^{asin}$ falling into each interval $P_i, i = 1, \ldots, s+1$ and compare the empirical distribution with the theoretical one. To be more precise, for $1 \leq i \leq s+1$ let

$$\mu'_m(P_i) = \mathbb{P}\left(S_n^{asin} \in P_i\right), \quad E_i = m \cdot \mu'_m(P_i),$$
$$\nu'_m(P_i) = \frac{|\{j : \ S_{n,j}^{asin} \in P_i, 1 \leq j \leq m\}|}{m}, \quad O_i = m \cdot \nu'_m(P_i),$$

where $\mathbb{P}\left(S_n^{asin} \in P_i\right) = F^{asin}(b) - F^{asin}(a)$ for $P_i = [a, b]$. Then statistics T^{asin} and d_{tv}^{asin} can be rewritten as

$$T^{asin} = m \cdot \sum_{\mathcal{A} \in \mathcal{P}_s} \frac{(\mu'_m(\mathcal{A}) - \nu'_m(\mathcal{A}))^2}{\mu'_m(\mathcal{A})}, \qquad d_{tv}^{asin} = \frac{1}{2} \sum_{\mathcal{A} \in \mathcal{P}_s} |\mu'_m(\mathcal{A}) - \nu'_m(\mathcal{A})|.$$

This technique was presented in [18] (for the total variation and few other distances) and this is how our implementation of the ASIN test [19] calculates the statistics.

We could also calculate just one p-value of the statistic $S_{n'}^{asin}$ for a longer sequence (say, for $n' = n \cdot m$) – *i.e.,* perform a *first level* test. However, as mentioned in Section 1, the *second level* approach produces more accurate results (roughly speaking, the accuracy is related to the ability, given a non-random PRNG, of recognizing its sequences as non-random, see details in [4]).

It is worth noting that the following approach can be applied when d_{tv}^{asin} or p_{χ^2} are *slightly* outside the acceptance region (*e.g.,* if $p_{\chi^2} \in (10^{-4}, 10^{-2})$, what suggests rejecting $\mathcal{H}_0$, but is not a *strong evidence*). Namely, double the length of the sequence, take a new output from the PRNG and apply the test again. Repeat the procedure (at most some predefined number of times) until the evidence is strong enough (*e.g.,* $p_{\chi^2} < 10^{-4}$) or $\mathcal{H}_0$ is accepted (*e.g.,* $p_{\chi^2} > 0.01$). This method, called *"automation of statistical tests on randomness"*, was proposed and analyzed in [29].

3.2. Error analysis

3.2.1. Bounding errors in approximating p-values in basic tests

In this subsection we will show a bound on the approximation error in (3). Recall that $F^{asin}(x) = \frac{2}{\pi} \arcsin \sqrt{t}$.

Lemma 3.2. *Fix a partition $\mathcal{P}_s, s \geq 2$ and an even $n \geq 2$. Let F_n be the cdf of the empirical distribution of S_n^{asin} under $\mathcal{H}_0$ (stating that the bits $B_1, \ldots, B_n$ were generated uniformly at random), i.e., $F_n(x) = \mathbb{P}\left(S_n^{asin} \leq x | \mathcal{H}_0\right)$. Then we have*

$$\sup_{x \in [0,1]} |F_n(x) - F^{asin}(x)| \leq \frac{C}{n}, \quad C = \frac{4}{3\pi}\left(2 - \frac{3}{2s}\right)\left(\frac{4s^2}{2s-1}\right)^{\frac{3}{2}}.$$

Proof. We will show that for fixed a and b such that $0 \leq a < b \leq 1$ we have

$$\left| \mathbb{P}\left(S_n^{asin} \in (a, b)\right) - \frac{1}{\pi} \int_a^b \frac{dt}{\sqrt{t(1-t)}} \right| \leq \frac{C}{n}.$$

Let us assume that $n = 2\mathfrak{n}$. Let $p_{2k,2\mathfrak{n}}$ denote the probability that during $2k$ steps in the first $2\mathfrak{n}$ steps the random walk was above the x-axis, *i.e.,* $p_{2k,2\mathfrak{n}} = \mathbb{P}\left(L_{2\mathfrak{n}} = 2k\right)$. The classical results on a simple random walk state that

$$p_{2k,2\mathfrak{n}} = \binom{2k}{k}\binom{2(\mathfrak{n}-k)}{\mathfrak{n}-k} 2^{-2\mathfrak{n}}. \tag{4}$$

The standard proof of Theorem 2.3 (see, *e.g.,* Chapter XII.8 in [30]) shows that $p_{2k,2\mathfrak{n}}$ converges to $d_{k,\mathfrak{n}} = \frac{1}{\pi\sqrt{k(\mathfrak{n}-k)}}$. In the following, we will bound the difference $|p_{2k,2\mathfrak{n}} - d_{k,\mathfrak{n}}|$. We will use a version of Stirling's formula stating that for each $\mathfrak{n}$ there exists $\theta_{\mathfrak{n}}$, $0 < \theta_{\mathfrak{n}} \leq 1$, such that

$$\mathfrak{n}! = \sqrt{2\pi\mathfrak{n}}\left(\frac{\mathfrak{n}}{e}\right)^{\mathfrak{n}} \exp\left(\frac{\theta_{\mathfrak{n}}}{12\mathfrak{n}}\right). \tag{5}$$

Plugging (5) into each factorial appearing in (4) we have

$$p_{2k,2\mathfrak{n}} = \frac{1}{\pi\sqrt{k(\mathfrak{n}-k)}} \exp\left(\frac{\theta_{2k} - 4\theta_k}{24k} + \frac{\theta_{2(\mathfrak{n}-k)} - 4\theta_{\mathfrak{n}-k}}{24(\mathfrak{n}-k)}\right).$$

Thus, we get

$$\frac{p_{2k,2\mathfrak{n}}}{d_{k,\mathfrak{n}}} \leq \exp\left(\frac{1}{24k} + \frac{1}{24(\mathfrak{n}-k)}\right) = \exp\left(\frac{\mathfrak{n}}{24k(\mathfrak{n}-k)}\right)$$

and

$$\frac{p_{2k,2\mathfrak{n}}}{d_{k,\mathfrak{n}}} \geq \exp\left(\frac{-4}{24k} + \frac{-4}{24(\mathfrak{n}-k)}\right) = \exp\left(-\frac{\mathfrak{n}}{6k(\mathfrak{n}-k)}\right).$$

For any x it holds that $1-e^{-x} \leq x$ and for $x \in [0, 1.25]$ we have that $e^x - 1 \leq 2x$. Note that $\frac{\mathfrak{n}}{24k(\mathfrak{n}-k)} \leq 1.25$, what is equivalent to $\mathfrak{n} \geq \frac{30k^2}{30k-1}$, what holds for any $0 < k < \mathfrak{n}$. Hence,

$$\begin{aligned} p_{2k,2\mathfrak{n}} - d_{k,\mathfrak{n}} &\leq d_{k,\mathfrak{n}}\left(\exp\left(\frac{\mathfrak{n}}{24k(\mathfrak{n}-k)}\right) - 1\right) \leq d_{k,\mathfrak{n}}\frac{\mathfrak{n}}{12k(\mathfrak{n}-k)}, \\ d_{k,\mathfrak{n}} - p_{2k,2\mathfrak{n}} &\leq d_{k,\mathfrak{n}}\left(1 - \exp\left(-\frac{\mathfrak{n}}{6k(\mathfrak{n}-k)}\right)\right) \leq d_{k,\mathfrak{n}}\frac{\mathfrak{n}}{6k(\mathfrak{n}-k)}, \end{aligned}$$

what implies

$$|p_{2k,2\mathfrak{n}} - d_{k,\mathfrak{n}}| \leq d_{k,\mathfrak{n}}\frac{\mathfrak{n}}{6k(\mathfrak{n}-k)} = \frac{\mathfrak{n}}{6\pi\left(k(\mathfrak{n}-k)\right)^{\frac{3}{2}}}.$$

Fix $\delta > 0$ and assume furthermore that $\delta \leq \frac{k}{\mathfrak{n}} \leq 1-\delta$. The function $k \mapsto (k(\mathfrak{n}-k))^{3/2}$ achieves the minimum value at the endpoints of the considered interval, thus

$$|p_{2k,2\mathfrak{n}} - d_{k,\mathfrak{n}}| \leq \frac{\mathfrak{n}}{6\pi\left(\delta\mathfrak{n}(\mathfrak{n}-\delta\mathfrak{n})\right)^{\frac{3}{2}}} = \frac{1}{6\pi\mathfrak{n}^2\left(\delta(1-\delta)\right)^{\frac{3}{2}}}.$$

We will estimate the approximation error in (3) in two steps. First, take two numbers a, b such that $\delta \leq a < b \leq 1-\delta$. We have

$$\left|\sum_{a\leq\frac{k}{\mathfrak{n}}\leq b} p_{2k,2\mathfrak{n}} - \sum_{a\leq\frac{k}{\mathfrak{n}}\leq b} d_{k,\mathfrak{n}}\right| \leq \sum_{a\leq\frac{k}{\mathfrak{n}}\leq b} |p_{2k,2n} - d_{k,\mathfrak{n}}| \leq \sum_{a\leq\frac{k}{\mathfrak{n}}\leq b} \frac{1}{6\pi\mathfrak{n}^2\left(\delta(1-\delta)\right)^{\frac{3}{2}}}$$

$$= \frac{\lceil b\mathfrak{n} - a\mathfrak{n}\rceil}{6\pi\mathfrak{n}^2\left(\delta(1-\delta)\right)^{\frac{3}{2}}} \leq \frac{b-a}{3\pi\mathfrak{n}\left(\delta(1-\delta)\right)^{\frac{3}{2}}} \leq \frac{1}{3\pi\mathfrak{n}\left(\delta(1-\delta)\right)^{\frac{3}{2}}} =: \eta.$$

The second kind of errors in probability estimates given by (3) is caused by approximating the sum by an integral. Let us consider an arbitrary function f differentiable in the interval (a, b). Split (a, b) into subintervals of length $\frac{1}{\mathfrak{n}}$ and let x_k be an arbitrary point in the interval containing $\frac{k}{\mathfrak{n}}$. Denote by M_k and

m_k the maximum and the minimum value of f on that interval, respectively. Using the Lagrange's mean value theorem we obtain

$$\left|\int_a^b f(x)dx - \sum_{a\le\frac{k}{\mathfrak{n}}\le b}\frac{1}{\mathfrak{n}}f(x_k)\right| \le \sum_{a\le\frac{k}{\mathfrak{n}}\le b}\frac{1}{\mathfrak{n}}(M_i-m_i) = \sum_{a\le\frac{k}{\mathfrak{n}}\le b}\frac{1}{\mathfrak{n}^2}|f'(\xi_i)| \le \sum_{a\le\frac{k}{\mathfrak{n}}\le b}\frac{1}{\mathfrak{n}^2}\sup_{a\le x\le b}|f'(x)|$$

$$= \frac{\lceil b\mathfrak{n}-a\mathfrak{n}\rceil}{\mathfrak{n}^2}\sup_{a\le x\le b}|f'(x)| \le \frac{2(b-a)}{\mathfrak{n}}\sup_{a\le x\le b}|f'(x)|.$$

For $f(x)=\frac{1}{\pi\sqrt{x(1-x)}}$ we have $f'(x)=\frac{2x-1}{2\pi(x(1-x))^{3/2}}$ and $\frac{1}{\mathfrak{n}}f\left(\frac{k}{\mathfrak{n}}\right)=d_{k,\mathfrak{n}}$. Hence, in the considered interval $(a,b)\subseteq(\delta,1-\delta)$ we have

$$\left|\int_a^b f(x)dx - \sum_{a\le\frac{k}{\mathfrak{n}}\le b} d_{k,\mathfrak{n}}\right| \le \frac{2}{\mathfrak{n}}\sup_{\delta<x<1-\delta}|f'(x)| = \frac{1-2\delta}{\pi\mathfrak{n}(\delta(1-\delta))^{\frac{3}{2}}} =: \kappa.$$

We also have

$$\left|\int_a^b f(x)dx - \sum_{a\le\frac{k}{\mathfrak{n}}\le b} p_{2k,2\mathfrak{n}}\right| \le \left|\int_a^b f(x)dx - \sum_{a\le\frac{k}{\mathfrak{n}}\le b} d_{k,\mathfrak{n}}\right| + \left|\sum_{a\le\frac{k}{\mathfrak{n}}\le b} d_{k,\mathfrak{n}} - \sum_{a\le\frac{k}{\mathfrak{n}}\le b} p_{2k,2\mathfrak{n}}\right| \le \eta+\kappa.$$

Taking $\delta=\frac{1}{2s}$ we obtain

$$\eta+\kappa = \frac{2}{3\pi\mathfrak{n}}\frac{2-3\delta}{(\delta(1-\delta))^{3/2}} = \frac{4}{3\pi\mathfrak{n}}\left(1-\frac{3}{4s}\right)\left(\frac{4s^2}{2s-1}\right)^{\frac{3}{2}} = \frac{\frac{1}{2}C}{\mathfrak{n}} = \frac{C}{n},$$

what justifies the approximation (3) for $\delta\le a<b\le 1-\delta$. To complete the analysis we need to investigate the errors "on the boundaries" of a unit interval, *i.e.,* for $(0,\delta)$ (and, by symmetry, for $(1-\delta,\delta)$). We get

$$\left|\int_0^\delta f(x)dx - \sum_{0\le\frac{k}{\mathfrak{n}}<\delta} p_{2k,2\mathfrak{n}}\right| = \left|\int_0^{\frac{1}{2}} f(x)dx - \int_\delta^{\frac{1}{2}} f(x)dx - \sum_{0\le\frac{k}{\mathfrak{n}}\le\frac{1}{2}} p_{2k,2\mathfrak{n}} + \sum_{\delta\le\frac{k}{\mathfrak{n}}\le\frac{1}{2}} p_{2k,2\mathfrak{n}}\right|$$

$$= \left|\frac{1}{2} - \int_\delta^{\frac{1}{2}} f(x)dx - \frac{1}{2} + \sum_{\delta\le\frac{k}{\mathfrak{n}}\le\frac{1}{2}} p_{2k,2\mathfrak{n}}\right| = \left|\int_\delta^{\frac{1}{2}} f(x)dx - \sum_{\delta\le\frac{k}{\mathfrak{n}}\le\frac{1}{2}} p_{2k,2\mathfrak{n}}\right| \le \frac{C}{n},$$

where the last inequality follows directly from the preceding calculations. □

Remark. Let $X_1, X_2, \ldots$ be zero-average i.i.d. random variables with $E|X_i|^3<\infty$. Denote $EX_i^2=\sigma^2$. The central limit theorem states that $\mathcal{N}$, a normal random variable $N(0,1)$ (denote its cdf by Φ), is the limiting distribution of $Y_n=\sum_{i=1}^n\frac{X_i}{\sigma\sqrt{n}}$ (denote its cdf by by F_n^Y). It means that *for large* n we can approximate Y_n by $\mathcal{N}$ and the approximation error is bounded by the Berry-Esseen inequality

$$\sup_x|F_n^Y(x)-\Phi(x)| \le \frac{C_0E|X_1|^3}{\sigma^3\sqrt{n}},$$

where C_0 is a positive constant (in original paper [31] it was shown that $C_0\le 7.59$, in [32] it was shown that $C_0\le 0.4785$). Lemma 3.2 is thus a Berry-Esseen type inequality for approximating S_n^{asin} by a random variable with cdf F^{asin}, tailored to our needs.

3.2.2. Reliability of the results from the second level test

Following [10], we say that a basic test (calculating $S_{n,j}^{asin}$) is *not reliable* if, due to approximation errors in the computations of p_j-values, the distribution of $p_j, j = 1, \ldots, m$ for truly random numbers is not uniform. We test the uniformity via T^{asin} and d_{tv}^{asin}. Since we compare two continuous distributions, some discretization needs to be applied. In our testing procedure we use a partition $\mathcal{P}_s$ for this, splitting the interval $[0, 1]$ into $(s + 1)$ intervals (*i.e.,* the bins). Lemma 3.2 states that a maximum error in the computation of p_j is bounded by $\frac{C}{n}$ (note that C implicitly depends on s). It means that a p_j-value that should belong to a given bin can be found in the neighboring ones only if the distance between p_j and one of the endpoints of a given bin is less than $\frac{C}{n}$. Thus, this is also the fraction of p_j-values that can be found in wrong bins. The maximum propagated deviation is twice the error (since most bins have two neighbors), *i.e.,*

$$\Delta = \frac{2C}{n}.$$

Under $\mathcal{H}_0$ the distribution of the numbers $p_j, j = 1, \ldots, m$ in the bins $1, \ldots, s+1$ is a multinomial distribution. Indeed, this is equivalent to throwing m balls independently into $s+1$ bins, where the probability of choosing first and last bin is $\frac{1}{2s}$ and $\frac{1}{s}$ for all remaining bins. The variance of the ratio of number of balls in bin $j \in \{1, s + 1\}$ is equal to $\frac{2s-1}{m4s^2}$, and for bin $j \in \{2, \ldots, s\}$ is equal to $\frac{s-1}{ms^2}$. We have $\sigma > \sqrt{\frac{s-1}{s^2m}}$, where σ is the expected statistical deviation of the ratio of p_j-values found in a given bin. We expect that the error in approximating p_j-values propagates into an additional deviation. If the deviation is smaller than the statistical deviation, *i.e.,* if

$$\Delta \leq \sigma, \tag{6}$$

then we say that the second level test is *reliable*. Note that the reliability of a test imposes a restriction on a relation between the length of a sequence used for each base test (*i.e.,* n) and the number of basic tests (m). Inequality (6) implies a lower bound on m, namely

$$m \leq (s - 1)\left(\frac{n}{2Cs}\right)^2. \tag{7}$$

4. The Flawed PRNG

In this section we present $\mathsf{Flawed}_{rng,N,\tau}$ – a family of PRNGs. The family depends on three parameters: *rng* (a PRNG, *e.g.,* the Mersenne Twister), N (a small integer, *e.g.,* $N = 30$) and $\tau \in [0, 1]$. $\mathsf{Flawed}_{rng,N,\tau}(seed)$ generates the output of length 2^N. For a fraction $1 - \tau$ of all possible seeds the output is the same as the output of $rng(seed)$. For the remaining fraction τ of seeds it outputs bits such that the corresponding walk spends exactly half of the time (2^{N-1} steps) above zero and exactly half of the time below zero.

4.1. Dyck Paths

To generate walks with the aforementioned property we will use Dyck paths, *i.e.,* walks starting and ending at 0 with the property that for each prefix the number of ones is not smaller than the number of zeros.

Definition 4.1. *Let* n *be an integer. A sequence of* $2\mathsf{n}$ *bits* $B_1, \ldots, B_{2\mathsf{n}}$ *is called a Dyck path if the corresponding walk* S_k *fulfills* $S_k = \sum_{i=1}^{k}(2B_i - 1) \geq 0, k = 1, \ldots, 2\mathsf{n} - 1$ *and* $S_{2\mathsf{n}} = 0$*. A set of all Dyck paths of length* $2\mathsf{n}$ *is denoted by* $\mathcal{D}_{2\mathsf{n}}$*.*

Thus, a Dyck path of length $2\mathsf{n}$ corresponds to a valid grouping of n pairs of parentheses. We have $|\mathcal{D}_{2\mathsf{n}}| = C_{\mathsf{n}} = \frac{1}{\mathsf{n}+1}\binom{2\mathsf{n}}{\mathsf{n}}$ (C_{n} is the n-th Catalan number).

We are interested in generating Dyck paths uniformly at random. To achieve this goal we will use the following three ingredients.

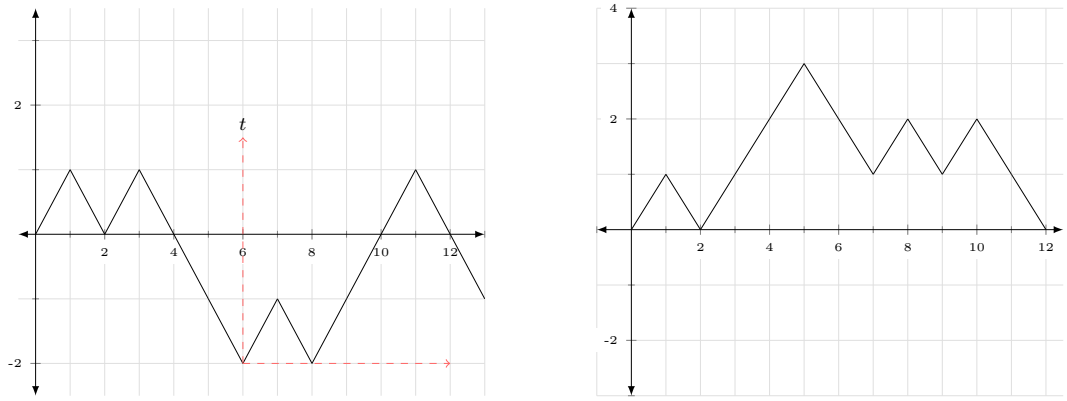

Figure 2: A path $I \in \mathcal{I}_{13}^{-1}$ (left) and the corresponding Dyck path $f_{\mathsf{Dyck}}(I)$ of length 12 (right).

(1) Walk sampling. Let $\mathcal{I}_{2\mathsf{n}+1}^{-1}$ be the set of sequences of bits $B_1, \ldots, B_{2\mathsf{n}+1}$ such that the corresponding walk S_k ends at -1, *i.e.,* $S_{2\mathsf{n}+1} = -1$. One can easily sample a sequence $I \in \mathcal{I}_{2\mathsf{n}+1}^{-1}$ uniformly at random – it is enough to make a random permutation of the vector of bits $(0, \ldots, 0, 1, \ldots, 1)$, consisting of $\mathsf{n}+1$ zeros and n ones.

(2) f_{Dyck} transformation. One can obtain a Dyck path of length $2\mathsf{n}$ from $I \in \mathcal{I}_{2\mathsf{n}+1}^{-1}$ using Algorithm 1.

Algorithm 1 $f_{\mathsf{Dyck}}(I)$

Input: $I = (B_1, \ldots, B_{2\mathsf{n}+1}) \in \mathcal{I}_{2\mathsf{n}+1}^{-1}$
Output: DyckPath – the Dyck path corresponding to I

1: $S_k = \sum_{i=1}^{k}(2B_i - 1)$ for $k = 1, \ldots, 2\mathsf{n}+1$
2: $t = \min(\operatorname{argmin}\{S_k : k \in \{1, \ldots, 2\mathsf{n}+1\}\})$
3: DyckPath $= (B_{t+1}, \ldots, B_{2\mathsf{n}+1}, B_1, \ldots, B_{t-1})$
4: **return** DyckPath

Observe that $f_{\mathsf{Dyck}}(I)$ transforms $I \in \mathcal{I}_{2\mathsf{n}+1}^{-1}$ into a Dyck path. This follows from simple observations:

1. I has exactly $\mathsf{n}+1$ zeros and n ones;
2. since $t = \min(\operatorname{argmin}\{S_k : k \in \{1, \ldots, 2\mathsf{n}+1\}\})$ then $B_t = 0$ and after B_t is removed then $f_{\mathsf{Dyck}}(I)$ has exactly n bits equal to 0 and n bits equal to 1;
3. from the definition of t (which enforces in particular that $B_{t+1} = 1$), the walk that corresponds to bits $(B_{t+1}, \ldots, B_{2\mathsf{n}+1}, B_1, \ldots, B_{t-1})$ cannot go below 0 (note that for $t = 2\mathsf{n}+1$ the walk is given by $(B_1, \ldots, B_{2\mathsf{n}})$).

An example of a f_{Dyck} transformation is presented in the Figure 2.

(3) The Cycle Lemma. The correspondence between the set $\mathcal{I}_{2\mathsf{n}+1}^{-1}$ and the set $\mathcal{D}_{2\mathsf{n}}$ is expressed by the Cycle Lemma (see, *e.g.,* [33]).

Lemma 4.2 (The Cycle Lemma)**.** *For any $I \in \mathcal{I}_{2\mathsf{n}+1}^{-1}$ the path $f_{Dyck}(I)$ is a Dyck path. Moreover, any Dyck path in $\mathcal{D}_{2\mathsf{n}}$ is the image of exactly $2\mathsf{n}+1$ paths in $\mathcal{I}_{2\mathsf{n}+1}^{-1}$.*

Thus, to obtain a *random sample of a Dyck path* of length $2\mathsf{n}$ one needs to run Algorithm 2. We use the following convention: for a pseudorandom generator *rng* and a seed *seed* we denote by $random = rng.Init(seed)$ an initialized object (with *rng* and *seed*), for which one can call $random.getBits()$ – a

function that (1) returns next bits of the generator (as many as required), and (2) the internal state of the *random* object is updated, so the following calls of the *getBits*() method return next bits.

Algorithm 2 sampleDyckPath($\mathsf{n}, rng, seed, b$)

Input: n – an integer
rng – a pseudorandom generator
$seed$ – a seed
b – a bit deciding if the output path should be over ($b = 0$) or under ($b = 1$) of the x-axis

Output: P – a sampled Dyck path of length $2\mathsf{n}$ (for $b = 0$) or a sampled Dyck path with its bits flipped ($x_i \to 1 - x_i$, for $b = 1$)

0: $random := rng.Init(seed)$
1: $x = (B_1, \ldots, B_{2\mathsf{n}+1})$ where $B_1 = \ldots = B_{\mathsf{n}+1} = 0$ and $B_{\mathsf{n}+2} = \ldots = B_{2\mathsf{n}+1} = 1$
2: $\sigma \leftarrow \mathsf{RandPerm}(2\mathsf{n}+1, random.getBits())$ – a random permutation σ of $2\mathsf{n}+1$ elements
3: $I = (B_{\sigma(1)}, \ldots, B_{\sigma(2\mathsf{n}+1)})$
4: $P = f_{\mathsf{Dyck}}(I)$
5: **if** $b = 1$ **then**
6: each bit x of P is flipped ($x \to 1 - x$)
7: **end if**
8: **return** P

We denote by $\mathsf{RandPerm}(n, seed)$ a function that returns a pseudorandom permutation of n elements, using *seed* as its seed.

4.2. The Flawed generator

As mentioned at the beginning of this section, the $\mathsf{Flawed}_{rng,N,\tau}(seed)$ generator – described as Algorithm 3 – generates 2^N bit sequences. It works exactly the same as the underlying generator for a fraction $(1 - \tau)$ of seeds (lines 1-2). For the remaining fraction τ of seeds (lines 3-8) the output is generated in the following way.

1. The first 2^{N-2} bits are exactly the same as the first 2^{N-2} bits of $rng(seed)$ (line 4).
2. The next 2^{N-2} bits $(z_{2^{N-2}+1}, \ldots, z_{2^{N-1}})$ are generated as follows:
   (a) a pseudorandom permutation π is generated (line 5),
   (b) the bit $z_{2^{N-2}+i}$ is set to be equal to $1 - z_{\pi(i)}$ (lines 6-8).
   As the result, there is the same number of zeros and ones in the first 2^{N-1} bits – these bits are denoted as $\mathbf{z}_L$ (*i.e.,* the corresponding walk is at zero at step 2^{N-1}).
3. The remaining 2^{N-1} bits (denoted as $\mathbf{z}_R$) in the block are obtained by calling $\mathsf{DyckPaths}(N, \mathbf{z}_L, rng, seed)$ (Algorithm 4). As the result, the whole block $\mathbf{z}_L\mathbf{z}_R$, concatenated blocks of $\mathbf{z}_L$ and $\mathbf{z}_R$, of 2^N output bits has the property that the corresponding walk spends the same number of steps above and below 0 (the description of DyckPaths is in Algorithm 4).

Algorithm 3 $\mathsf{Flawed}_{rng,N,\tau}(seed)$

Input: N – an integer
rng – a pseudorandom generator
$seed$ – a seed
$\tau \in (0,1)$ – a parameter deciding which fraction of seeds returns flawed output
Output: $(z_1, \ldots, z_{2^N})$ – generated bits

0: $random := rng.Init(seed)$
1: **if** $seed \neq 0 \bmod \lceil 1/\tau \rceil$ **then**
2: **return** $random.getBits()$
3: **else**
4: $(z_1, \ldots z_{2^{N-2}}) \leftarrow random.getBits()$
5: $\pi \leftarrow \mathsf{RandPerm}(2^{N-2}, random.getBits())$
6: **for** $i = 1$ to 2^{N-2} **do**
7: $z_{2^{N-2}+i} := 1 - z_{\pi(i)}$
8: **end for**
9: $seed := random.getBits()$
10: $(z_{2^{N-1}+1}, \ldots, z_{2^N}) \leftarrow \mathsf{DyckPaths}(N, (z_1, \ldots, z_{2^{N-1}}), rng, seed)$
11: **return** $(z_1, \ldots, z_{2^N})$
12: **end if**

Example 4.3 (Flawed)**.** *Let $N = 5$ and let us assume that the seed used in Algorithm 3 has such a value that lines 4-11 are executed. Let the result of line 4 be $(z_1, \ldots, z_8) = (1,1,0,0,0,0,0,1) \leftarrow rng(seed)$ and line 5 returns a permutation*

$$\pi = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 7 & 3 & 6 & 1 & 4 & 8 & 2 & 5 \end{pmatrix}.$$

Then the bits computed in lines 6-8 are

$$(z_9, z_{10}, z_{11}, z_{12}, z_{13}, z_{14}, z_{15}, z_{16}) = (1-z_7, 1-z_3, 1-z_6, 1-z_1, 1-z_4, 1-z_8, 1-z_2, 1-z_5) = (1,1,1,0,1,0,0,1).$$

Then bits $\mathbf{z_L} = (z_1, \ldots, z_8, z_9, \ldots, z_{16})$ are used as an input to function $\mathsf{DyckPaths}$ *in line 10. The example is continued as Example 4.4.*

Let us assume that Algorithm 3 (Flawed) has generated bits $\mathbf{z}_L = (z_1, \ldots, z_{2^{N-1}})$ by executing lines $4-8$. Then the corresponding random walk spent $T_U(\mathbf{z}_L)$ time under the x-axis and $T_O(\mathbf{z}_L)$ time over the x-axis ($T_U(\mathbf{z}_L) + T_O(\mathbf{z}_L) = 2^{N-1}$). The goal of the procedure $\mathsf{DyckPaths}(N, \mathbf{z}_L, rng, seed)$ is to generate bits $\mathbf{z}_R = (z_{2^{N-1}+1}, \ldots, z_{2^N})$ in such a way that $T_U(\mathbf{z}_R) = T_O(\mathbf{z}_L)$ and $T_O(\mathbf{z}_R) = T_U(\mathbf{z}_L)$. Then if one concatenates sequences $\mathbf{z}_L$ and $\mathbf{z}_R$, for the corresponding random walk it holds that $T_U(\mathbf{z}_L\mathbf{z}_R) = T_O(\mathbf{z}_L\mathbf{z}_R)$.

The following is an informal explanation of the procedure $\mathsf{DyckPaths}(N, \mathbf{z}_L, rng, seed)$ (the formal description is provided by Algorithm 4).

1. The walk corresponding to $\mathbf{z}_L$ is given by $S_0 := 0, S_k = \sum_{i=1}^{k}(2z_i - 1), k = 1, \ldots, 2^{N-1}$.
2. The sequence D_k is defined as: $D_k = \mathbb{1}(S_k > 0 \vee S_{k-1} > 0)$, for $k = 1, \ldots, 2^{N-1}$.
3. The set of points where the walk changes its sign is defined as $R = \{i | i = 1, ..., 2^{N-1} - 1, D_{i+1} \neq D_i\} \cup \{2^{N-1}\}$.
4. Elements of R are sorted in increasing order (obtaining $(r_1, \ldots, r_w)$).
5. The sequence $\{l_i\}$ is defined as: $l_1 := 1$, and the next "left-ends" as $l_i = r_{i-1} + 1$ (for $i = 2, \ldots, w$).
6. The set O_i is defined as $O_i := \{l_i, l_1 + 1, \ldots, r_i\}$, for $i = 1, \ldots, w$.
7. Bits $2^{N-1} + 1, \ldots, 2^N$ ($\mathbf{z}_R$) are chosen so that the whole walk spends the same number of steps over and under the x-axis. Dyck's paths are generated[1]: $DP_i = \mathsf{sampleDyckPath}(|O_i|/2, rng, seed, D_{r_i})$, for $i = 1, \ldots, w$ and $seed = random.getBits()$, so consecutive DP_i are computed with different seeds.

[1]The definitions of l_i and r_i imply that $|O_i|$ is even, $i = 1, \ldots, w$

8. A relative ordering of the paths (generated in the previous step) is obtained from a permutation $\rho \leftarrow \mathsf{RandPerm}(w, rng(seed))$.
9. The resulting bits are obtained by concatenating permuted Dyck's paths.

Example 4.4 (DyckPaths)**.** *Let input to* DyckPaths *be* $z_{\mathbf{L}} = (z_1, \ldots, z_{16}) = (1,1,0,0,0,0,0,1,1,1,1,0,1,0,0,1)$. *Then* $(r_1, r_2, r_3, r_4) = (4, 10, 14, 16), (l_1, l_2, l_3, l_4) = (1, 5, 11, 15)$ *and thus* $O_1 = \{1,2,3,4\}, O_2 = \{5,6,7,8,9,10\}, O_3 = \{11,12,13,14\}, O_4 = \{15,16\}$.

Let the output of sampleDyckPath *(called in lines 7-9 of* DyckPaths*) be* $DP_1 = (0,1,0,1)$, $DP_2 = (1,1,0,1,0,0)$, $DP_3 = (0,0,1,1)$, $DP_4 = (1,0)$.

Let

$$\rho = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 2 & 1 & 4 \end{pmatrix}.$$

Then $(z_{17}, \ldots, z_{32}) = DP_{\rho(1)} DP_{\rho(2)} DP_{\rho(3)} DP_{\rho(4)} = (0,0,1,1,1,1,0,1,0,0,0,1,0,1,1,0)$.

Algorithm 4 $\mathsf{DyckPaths}(N, (z_1, \ldots, z_{2^{N-1}}), rng, seed)$

Input: N – an integer
$(z_1, \ldots, z_{2^{N-1}})$ – a sequence of input bits
rng – a pseudorandom generator
$seed$ – a seed

Output: $(z_{2^{N-1}+1}, \ldots, z_{2^N})$ – generated bits

0: $random := rng.Init(seed)$
1: $S_0 := 0, \quad S_k := \sum_{i=1}^{k} (2z_i - 1), \quad k = 1, \ldots, 2^{N-1}$
2: $D_k := \mathbb{1}(S_k > 0 \vee S_{k-1} > 0), k = 1, \ldots, 2^{N-1}$
3: $R := \{i : D_{i+1} \neq D_i\} \cup \{2^{N-1}\}, i = 1, \ldots, 2^{N-1} - 1$
4: Let $(r_1, \ldots, r_w)$ be the sorted sequence of elements of R
5: $l_i = \begin{cases} 1 & i = 1 \\ r_{i-1} + 1 & i = 2, \ldots, w \end{cases}$
6: $O_i := \{l_i, \ldots, r_i\}$ for $i = 1, \ldots, w$
7: **for** $i = 1, \ldots, w$ **do**
8: $\quad seed = random.getBits()$
9: $\quad DP_i = \mathsf{sampleDyckPath}(|O_i|/2, rng, seed, D_{r_i})$
10: **end for**
11: $\rho \leftarrow \mathsf{RandPerm}(w, random.getBits())$
12: $(z_{2^{N-1}+1}, \ldots, z_{2^N}) = DP_{\rho(i)} \ldots DP_{\rho(w)}$
13: **return** $(z_{2^{N-1}+1}, \ldots, z_{2^N})$

Ten sample trajectories of the Flawed generator (all generated by the *Dyck path*-based part of Algorithm 3) are depicted in Figure 3 (the instance of $\mathsf{Flawed}_{rng,N,\tau}$ was initialized with the following parameters: $N = 18$, rng – the Mersenne Twister).

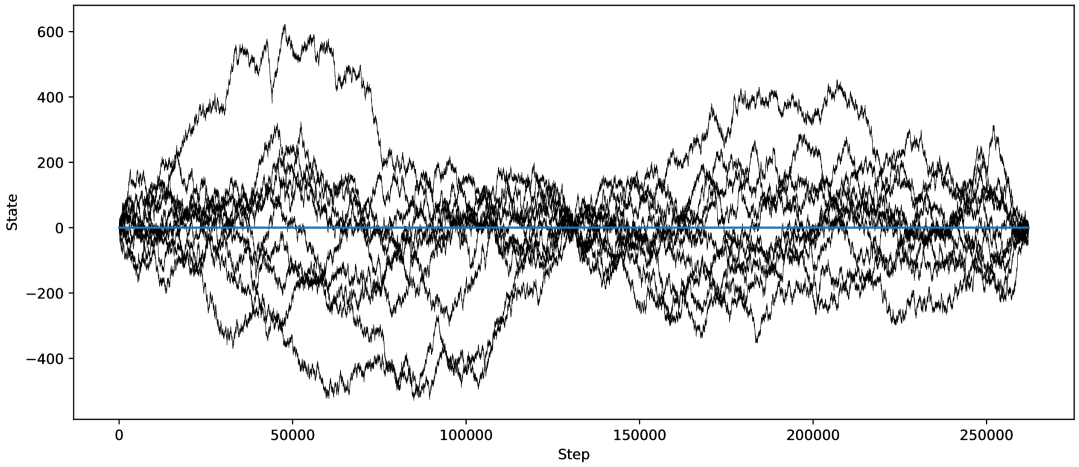

Figure 3: 10 trajectories of length 2^{18} produced by the Flawed generator for ten seeds of form $seed = k \cdot \lceil 1/\tau \rceil\}, k = 0, \ldots, 9$, *i.e.,* each path resulted from lines 4-10 of Algorithm 3

Discussion on the influence of the parameter τ of the Flawed PRNG on the statistic T^{asin}. Recall that in the Algorithm 3 the parameter τ corresponds to a fraction of simulations which are exactly half of the time above and half of the time below the x-axis, *i.e.,* we have $S_{n,j}^{asin} = 0.5$ for $\lfloor \tau m \rfloor$ simulations. Note that the p_j-value is then also equal to 0.5. The remaining $\lceil (1-\tau)m \rceil$ simulations come from the *rng*. Let us assume that the *rng* returns truly random numbers. Concerning the statistic T^{asin}, we have $E_1 = E_{s+1} = \frac{m}{2s}$ and $E_i = \frac{m}{s}, i = 2, \ldots, s$. Set $r := \lceil \frac{s}{2} \rceil + 1$. For an "ideal" *rng* we would have $O_1 = O_{s+1} = \frac{m(1-\tau)}{2s}, O_k = \frac{m(1-\tau)}{s}, k \in \{2, \ldots, s\} \setminus \{r\}$ and $O_r = \frac{m(1-\tau)}{s} + \tau m$. Thus,

$$T^{asin} = \sum_{i=1}^{s+1} \frac{(O_i - E_i)^2}{E_i} = (s-2)\frac{(\frac{m(1-\tau)}{s} - \frac{m}{s})^2}{\frac{m}{s}} + 2\frac{(\frac{m(1-\tau)}{2s} - \frac{m}{2s})^2}{\frac{m}{2s}} + \frac{(\frac{m(1-\tau)}{s} + \tau m - \frac{m}{s})^2}{\frac{m}{s}}$$

$$= (s-2)\frac{m}{s}\tau^2 + \frac{m}{s}\tau^2 + \frac{m}{s}(\tau(s-1))^2 = m\tau^2(s-1). \quad (8)$$

5. Experimental results

In this section we briefly report our experimental results of testing some widely used PRNGs implemented in standard libraries in various programming languages. We have applied the ASIN test to different generators including the implementations of the standard C/C++ linear congruential generators, the standard generator RAND from the GNU C Library, the Mersenne Twister, the Minstd and the Combined Multiple Recursive Generator (CMRG) from Example 4 in [34].

As our last example we show the results of testing the Flawed generator. Flawed is identified by our ASIN test as non-random, whereas it passed many other tests, including all closely related procedures (`swalk_RandomWalk1` test from `TestU01` with statistics: H, M, J, R, C, see Table 2).

Each considered PRNG was tested by generating $m = 10000$ sequences of length $n \in \{2^{26}, 2^{30}, 2^{34}\}$, using the partition $\mathcal{P}_{40}^{asin}$, *i.e.,* $s = 40$. For these parameters our second level test is reliable (see Section 3.2.2) – σ, the expected statistical deviation of the ratio of p_j values found in a given bin is greater than $\sqrt{\frac{s-1}{s^2 m}} = 0.0015$, what significantly exceeds the maximum propagated error $\Delta = \frac{2C}{n}$, *i.e.,* the inequality (6) holds. Note that for $s = 40$ the inequality (7) yields:

- $\Delta = 7.0703 \cdot 10^{-8}, m \leq 4.8760 \cdot 10^{12}$ for $n = 2^{34}$,
- $\Delta = 0.00000113, m \leq 1.9047 \cdot 10^{10}$ for $n = 2^{30}$, and

Table 1: Results of the ASIN test for several generators with parameters $m = 10000, n = 2^{34}, s = 40$.

	d_{tv}^{asin}	p_{χ^2}
MS Visual C++	**0.2093**	**0.0000**
GNU C	0.0255	0.2389
Minstd 48271	**0.2089**	**0.0000**
MT19937-64	0.0252	0.2523

Table 2: Results of the statistic T^{asin} with corresponding p-values and `swalk_RandomWalk1` statistics from `TestU01` for $n = 2^{26}$.

	ASIN		`TestU01`				
PRNG\Test	T^{asin}	p_{χ^2}	Statistic H	Statistic M	Statistic J	Statistic R	Statistic C
MS Visual C++	61.86	0.0148	0.2700	0.0900	0.6300	0.4200	0.8000
GNU C	39.93	0.4731	0.1600	0.9800	0.1100	0.1900	0.4900
Minstd 48271	65.88	0.0096	0.0090	0.1400	0.4900	0.0700	0.0044
MT19937-64	45.47	0.2548	0.0800	0.1000	0.4200	0.9700	0.3500
$\mathsf{Flawed}_{\mathrm{MT19937-64,26,1/66}}$	94.08	**0.0000**	**0.0000**	0.2200	**0.0000**	0.3900	0.3800

- $\Delta = 0.000018$, $m \leq 7.44027 \cdot 10^7$ for $n = 2^{26}$.

In the experiments we used our custom implementations of tested PRNGs (except the Mersenne Twister). We used 64-bit version of C++11 implementation of the Mersenne Twister, *i.e.,* the class `std::mt19937_64`, which is, however, known to have some problems [35]. The generators were initialized with random seeds from `http://www.random.org` [36] and each sequence was generated using different seed.

The results are presented in Table 1. The values indicating that $\mathcal{H}_0$ should be rejected (w.r.t. significance level $\alpha = 0.0001$) are **bolded**. For p_{χ^2} these are simply the values smaller or equal to α. Concerning the values of d_{tv}^{asin}, Lemma 3.1 implies that for $\varepsilon \leq \sqrt{20 \cdot 41/10000} \leq 0.2862$ we have $\mathbb{P}\left(d_{tv}^{asin} > \varepsilon/2\right) \leq 3\exp\left(-400\varepsilon^2\right)$. It can be checked that $3\exp(-400\varepsilon^2) \leq 0.0001$ for $\varepsilon \leq 0.1605$, in other words

$$\mathbb{P}\left(d_{tv}^{asin} > 0.0802\right) \leq 0.0001,$$

i.e., we reject $\mathcal{H}_0$ if the value of d_{tv}^{asin} is larger than 0.0802.

We have also calculated the `swalk_RandomWalk1` statistics from `TestU01` for 10000 sequences of length 2^{26} of each PRNG. The following parameters for `swalk_RandomWalk1` were used: $N = 1, n = 10000, r = 0, s = 32, L0 = L1 = 2^{26}$. The results are given in Table 2 (including the Flawed generator described in Section 4). For each Statistic H, M, J, R and C, the corresponding p-values were obtained using the *chi-square* statistics. We also include the values of the statistic T^{asin} and corresponding p-values for each generator. Note that for these parameters an *"ideal"* rng (see paragraph *Discussion on the influence of the parameter τ of the Flawed PRNG on the statistic T^{asin}* in Section 4.2 and equation (8)) would yield $T^{asin} = m\tau^2(s-1) = \frac{10000 \cdot 39}{66^2} = 89.5316$. As can be seen in Table 2, only the result $\mathsf{Flawed}_{\mathrm{MT19937-64,26,1/66}}$ is close to this value.

Our ASIN test would reject the MS Visual C++ PRNG and the Minstd with a multiplier 48271 (The Minstd with a multiplier 16807 gave similar results - not reported here) as good PRNGs. Note that this is indicated by both p_{χ^2} and the value of d_{tv}^{asin}. We also conducted the experiments for the procedure `rand` from the standard library in the Borland C/C++ (not included here). The outcomes are very akin to those for a standard PRNG in the MS Visual C++. Note that for $n = 2^{26}$ none of the p-values calculated by the `swalk_RandomWalk1` from `TestU01` suggests rejecting the hypothesis that the MS Visual C++ PRNG is good. It is worth mentioning that the MS Visual C++ PRNG passes the NIST Test Suite [8], as pointed out in [18]. Minstd, despite its weaknesses, became a part the C++11 standard library. It is implemented by the classes `std::minstd_rand0` (with the multiplier 16807) and `std::minstd_rand` (with the multiplier 48271). Concerning the GNU C and the MT19937-64 – as can be seen in both Table 1 and Table 2 – they

can be both considered as good. It is worth mentioning that the results for the CMRG generator (not reported here) were similar to those for the MT19937-64.

The open source code of our implementation is publicly available, see [19] (it includes the Flawed PRNG as well as the *Law of Iterated Logarithm test* from [18]).

5.1. Results of `TestU01` for Flawed

We have run several general-purpose tests against the $\mathsf{Flawed}_{MT,30,1/66}$ generator. For `SmallCrush` all 15 out of 15 tests were passed. For the Mersenne Twister (MT) and the $\mathsf{Flawed}_{MT,30,1/66}$ we run `BigCrush`. Tests for which generators failed are presented in the Table 3. In addition, we have run FIPS_140_2 tests (issued by NIST, included in `TestU01`) on $\mathsf{Flawed}_{MT,26,1/66}$, Mersenne Twister and MS Visual C++. All the tests for all three PRNGs were passed.

Table 3: Tests from `BigCrush` which failed. The first column tno is the test number, for the p-value, ε is a value such that $\varepsilon < 1.0e-15$.

Mersenne Twister			
tno	test name	parameters	p-value
74	RandomWalk1 R	$L = 50, r = 0$	$6.1e-4$
80	LinearComp	$r = 0$	$1 - \varepsilon$
81	LinearComp	$r = 29$	$1 - \varepsilon$

$\mathsf{Flawed}_{MT,30,1/66}$			
tno	test name	parameters	p-value
80	LinearComp	$r = 0$	$1 - \varepsilon$
81	LinearComp	$r = 29$	$1 - \varepsilon$
88	PeriodsInStrings	$r = 0$	$1.1e-4$
89	PeriodsInStrings	$r = 20$	$1.3e-19$
102	Run of bits	$r = 27$	$7.1e-4$

6. Notes on Takashima's method for testing PRNGs and the arcsine test implementation from TestU01

The idea of using the arcsine law for developing statistical tests for an empirical evaluation of PRNGs was formerly proposed by Takashima in [37, 38, 39]. In this series of articles, test statistics based on the arcsine law were applied for assessing the randomness of the output of maximum-length linearly recurring sequences (m-sequences in short). The experimental results presented there clearly show that the bits produced by this family of PRNGs are biased. Besides revealing the weakness of m-sequences, these outcomes have also proved that Takashima's tests are effective methods, worth applying in practice.

The approach introduced in [37, 38] can be briefly described as follows. After an initialization of a PRNG, a sequence of $2\mathfrak{n}m$ bits is generated and divided into m subsequences of length $n = 2\mathfrak{n}$. Then, each subsequence is used for constructing a random walk. For each of these m sample random walks, the value of a test statistic based on the arcsine law is calculated. The investigated statistic, called in [37, 38] the *sojourn time* – denote them by $t_n^j, j = 0, \ldots, m-1$ – is the time spent by a random walk above the x-axis. From m realizations of this statistic an empirical distribution of the sojourn time $f_{2i} = |\{j : t_n^j = 2i\}|$, $i = 0, \ldots, \mathfrak{n}$ is then derived and compared with its theoretical distribution via a chi-square test. The whole procedure is repeated $\lambda \geq 1$ times, yielding a set of χ^2 test statistics' values $\{\chi_k^2\}, k = 0, \ldots, \lambda - 1$. The final step of the Takashima's testing method is to count the number of χ_k^2 values falling between 90-th and 95-th percentile and those bigger than 95-th percentile of a respective χ^2 distribution. These two counts are then the basis for deciding if $\mathcal{H}_0$ should be rejected. Note that for $\lambda > 1$ this is a third level test, which in general is *not reliable*, as shown in [3].

The author in [38] considers also a slightly modified variant of the procedure, where the chi-square test is combined with the Kolomogorov-Smirnov test. Another method, presented in [39], exploits the relations between the sojourn time and the last visit time for one-dimensional random walks.

It is worth noting that in our simulations we used binary sequences of length at least $n = 2^{26}$. Thus, a direct application of the Takashima's methods from [37, 38] would require large amount of additional memory to store the values of $f_{2i}, i = 0, \ldots, \mathfrak{n}$.

As the arcsine law based statistical tests were proven to be useful in detecting flaws of some PRNGs, such procedures were implemented in the `TestU01` library (see [7]). This tool, developed by L'Ecuyer and Simard, provides a big variety of functions for empirical examining of PRNGs. One of the test modules, `swalk`, contains a procedure `swalk_RandomWalk1`, which calculates a bunch of test statistics for a sample of m random walks constructed from chosen bits of generated binary sequences. Among them, there is the Statistic J, which implements the test based on the arcsine law. This procedure is similar to ours. Namely, m calculated values of the test statistic are grouped according to some partition and their empirical distribution is compared with the theoretical one by means of the chi-square test. The main difference is that in our testing method the partition size s is a parameter chosen by the user, whereas the partition used by `swalk_RandomWalk1` is calculated automatically, depending on the tested sequence. Moreover, we provide bounds on approximation errors in the computations of p-values (a Berry-Esseen type inequality), assuring the *reliability* of the whole testing procedure.

7. Conclusions

In this paper we analyzed a method for testing PRNGs based on the arcsine law for random walks. Our procedure is a second level statistical test. We also provided a detailed error analysis of the proposed method. The approximation errors in the calculation of p-values are bounded by a Berry-Esseen type inequality, what allows to control the overall error, assuring the *reliability* of the test. We evaluate the quality of PRNGs via the chi-square statistics as well as by calculating a statistical distance (the total variation distance) between the empirical distribution of the considered characteristic for generated pseudorandom output and its theoretical distribution for truly random binary sequences.

The experimental results presented in this paper show that our testing procedure can be used for detecting weaknesses in many common PRNGs' implementations. Likewise the *Law of Iterated Logarithm test* from [18], the ASIN test has also revealed some flaws and regularities in generated sequences not necessarily being identified by other current state of the art tools like the NIST SP800-22 Testing Suite or `TestU01`. Thus, these kind of testing techniques seem to be very promising, as they allow also for recognition of different kinds of deviations from those detected by existing tools. Nevertheless, like other statistical tests, the ASIN test is not universal and encompasses only one from an immense range of characteristics of random bit strings and does not capture all known flaws. Therefore, the testing procedures relying on properties of random walks like the ASIN test should be used along with other tests for more careful assessment of pseudorandom generators. This issue is well depicted by the provided example of obviously non-random generator Flawed for which the LIL test has failed to detect its weaknesses, but the ASIN test has turned out to be very sensitive for that kind of deviations. Hence, an important line of further research is to develop another novel tests utilizing various properties of random walks. Such tests, when combined together, should be capable of detecting more hidden dependencies between the consecutive bits in the sequences generated by PRNGs. This could lead to designing more robust test suites for evaluating the quality of random numbers generated by the new implementations of PRNGs as well as those being already in use, especially for cryptographic purposes.

Acknowledgements

We would like to thank the anonymous reviewers whose suggestions and insightful comments helped significantly improve and clarify this manuscript. In particular we thank one of the reviewers for pointing out the article [4] on second level tests.

References

[1] G. Marsaglia, The Structure of Linear Congruential Sequences, in: S. Zaremba (Ed.), Applications of Number Theory to Numerical Analysis, Academic Press, 1972, pp. 249–285.

[2] D. E. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3rd Edition, Addison-Wesley Pub. Co, 1997.

[3] P. L'Ecuyer, Testing Random Number Generators, in: Proceedings of the 24th Conference on Winter Simulation, WSC '92, ACM, New York, NY, USA, 1992, pp. 305–313.
[4] F. Pareschi, R. Rovatti, G. Setti, Second-level NIST Randomness Tests for Improving Test Reliability, in: 2007 IEEE International Symposium on Circuits and Systems, 2007, pp. 1437–1440. doi:10.1109/ISCAS.2007.378572.
[5] R. G. Brown, D. Eddelbuettel, D. Bauer, Dieharder: A Random Number Test Suite, http://www.phy.duke.edu/~rgb/General/dieharder.php (Accessed: 2019-07-03).
[6] P. L'Ecuyer, R. Simard, TestU01: A C Library for Empirical Testing of Random Number Generators, ACM Trans. Math. Softw. 33 (4) (2007) 22:1–22:40. doi:10.1145/1268776.1268777.
[7] P. L'Ecuyer, R. Simard, TestU01: A Software Library in ANSI C for Empirical Testing of Random Number Generators. Software user's guide, version of May 16, 2013, http://simul.iro.umontreal.ca/testu01/tu01.html/ (2013).
[8] NIST.gov - Computer Security Division - Computer Security Resource Center, NIST Test Suite, https://csrc.nist.gov/projects/random-bit-generation/ (Accessed: 2019-07-03).
[9] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, Tech. Rep. Rev. 1a, NIST (2010).
[10] F. Pareschi, R. Rovatti, G. Setti, Second-level testing revisited and applications to NIST SP800-22, in: 2007 18th European Conference on Circuit Theory and Design, IEEE, 2007, pp. 627–630.
[11] P. L'Ecuyer, R. Simard, S. Wegenkittl, Sparse Serial Tests of Uniformity for Random Number Generators, SIAM J. Sci. Comput. 24 (2) (2002) 652–668.
[12] M. Matsumoto, T. Nishimura, A Nonempirical Test on the Weight of Pseudorandom Number Generators, in: K.-T. Fang, H. Niederreiter, F. J. Hickernell (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2000, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 381–395.
[13] P. C. Leopardi, Testing the Tests: Using Random Number Generators to Improve Empirical Tests, in: P. L' Ecuyer, A. B. Owen (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2008, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 501–512.
[14] C. Kim, G. H. Choe, D. H. Kim, Tests of randomness by the gambler's ruin algorithm, Applied Mathematics and Computation 199 (1) (2008) 195–210.
[15] H. Ekkehard, A. Grønvik, Re-seeding invalidates tests of random number generators, Applied Mathematics and Computation 217 (1) (2010) 339–346.
[16] P. Lorek, M. Slowik, F. Zagórski, Statistical Testing of PRNG: Generalized Gambler's Ruin Problem, in: J. Blömer, I. S. Kotsireas, T. Kutsia, D. E. Simos (Eds.), Mathematical Aspects of Computer and Information Sciences - 7th International Conference, MACIS 2017, Vienna, Austria, November 15-17, 2017, Proceedings, Vol. 10693 of Lecture Notes in Computer Science, Springer, 2017, pp. 425–437. doi:10.1007/978-3-319-72453-9.
[17] W. Feller, An Introduction to Probability Theory and Its Applications, Volume 1, 3rd Edition, John Wiley & Sons, 1968.
[18] Y. Wang, T. Nicol, On statistical distance based testing of pseudo random sequences and experiments with PHP and Debian OpenSSL, Computers & Security 53 (2015) 44–64.
[19] P. Lorek, G. Łoś, F. Zagórski, K. Gotfryd, PRNG_Arcsine_test: Empirical tests for PRNGs based on the arcsine law. GitHub repository, https://github.com/lorek/PRNG_Arcsine_test (2019).
[20] S. Asmussen, P. W. Glynn, Stochastic Simulation: Algorithms and Analysis, Vol. 57 of Stochastic Modelling and Applied Probability, Springer-Verlag New York, 2007.
[21] D. P. Kroese, T. Taimre, Z. I. Botev, Handbook of Monte Carlo Methods, Wiley Series in Probability and Statistics, John Wiley & Sons, Hoboken, NJ, USA, 2011.
[22] P. L'Ecuyer, History of uniform random number generation, in: 2017 Winter Simulation Conference (WSC), IEEE, 2017, pp. 202–230.
[23] H. Niederreiter, Quasi-Monte Carlo methods and pseudo-random numbers, Bulletin of the American Mathematical Society 84 (6) (1978) 957–1041.
[24] M. Denker, W. A. Woyczynski, Introductory Statistics and Random Phenomena. Uncertainty, Complexity, and Chaotic Behavior in Engineering and Science, Birkhäuser Boston, 1998.
[25] A. Gut, Probability: A Graduate Course, Springer Texts in Statistics, Springer-Verlag New York, 2005.
[26] A. Khintchine, Über einen Satz der Wahrscheinlichkeitsrechnung, Fundamenta Mathematicae 6 (1) (1924) 9–20.
[27] L. Devroye, The Equivalence of Weak, Strong and Complete Convergence in L_1 for Kernel Density Estimates, The Annals of Statistics 11 (3) (1983) 896–904.
[28] D. Berend, A. Kontorovich, On the Convergence of the Empirical Distribution, https://arxiv.org/abs/1205.6711v2 (2012).
[29] H. Haramoto, Automation of Statistical Tests on Randomness to Obtain Clearer Conclusion, in: P. L' Ecuyer, A. B. Owen (Eds.), Monte Carlo and Quasi-Monte Carlo Methods 2008, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 411–421.
[30] W. Feller, An Introduction to Probability Theory and Its Applications, Volume 2, 2nd Edition, John Wiley & Sons, 1971.
[31] C. G. Esseen, On the Liapunoff limit of error in the theory of probability, Ark. Mat. Astr. Fysik 28A (2) (1942) 1–19.
[32] I. S. Tyurin, Refinement of the upper bounds of the constants in Lyapunov's theorem, Russian Mathematical Surveys 65 (3) (2010) 586–588.
[33] A. Dvoretzky, T. Motzkin, A problem of arrangements, Duke Mathematical Journal 14 (2) (1947) 305–313.
[34] P. L'Ecuyer, Combined Multiple Recursive Random Number Generators, Oper. Res. 44 (5) (1996) 816–822.
[35] S. Harase, Conversion of Mersenne Twister to double-precision floating-point numbers, https://arxiv.org/abs/1708.06018 (2017).

[36] M. Haahr, RANDOM.ORG: True Random Number Service, `https://www.random.org` (Accessed: 2019-07-03).
[37] K. Takashima, Sojourn time test for maximum-length linearly recurring sequences with characteristic primitive trinomials, Journal of the Japanese Society of Computational Statistics 7 (1994) 77–87.
[38] K. Takashima, Sojourn time test of m-Sequences with characteristic pentanomials, Journal of the Japanese Society of Computational Statistics 8 (1995) 37–46.
[39] K. Takashima, Last visit time tests for pseudorandom numbers, Journal of the Japanese Society of Computational Statistics 9 (1996) 1–14.