Applied Mathematics and Computation 426 (2022) 127116

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

A new approach to analyze the independence of statistical tests of randomness

Elena Almaraz Luengo[a], Marcos Brian Leiva Cerna[a], Luis Javier García Villalba[a,*], Julio Hernandez-Castro[b]

[a] Group of Analysis, Security and Systems (GASS), Universidad Complutense de Madrid Spain
[b] School of Computing, University of Kent, UK

ARTICLE INFO

Article history:
Received 10 November 2021
Revised 17 March 2022
Accepted 20 March 2022
Available online 2 April 2022

Keywords:
Cryptography
Dieharder
Generators
Hypothesis testing
Mutual information
Pearson's correlation
Pseudo-random numbers
Random numbers
TestU01
TufTest

ABSTRACT

One of the fundamental aspects when working with batteries of statistic tests is that they should be as efficient as possible, i.e. that they should check the properties and do so in a reasonable computational time. This assumes that there are no tests that are checking the same properties, i.e. that they are not correlated. One of the most commonly used measures to verify the interrelation between variables is the Pearson's correlation. In this case, linear dependencies are checked, but it may be interesting to verify other types of non-linear relationships between variables. For this purpose, mutual information has recently been proposed, which measures how much information, on average, one random variable provides to another. In this work we analyze some well-known batteries by using correlation analysis and mutual information approaches.

1. Introduction

The so-called Internet of Things (IoT) is one of the fields with the most potential for development nowadays. It consists of interconnecting a large number of small devices with reduced resources and consumption so that they can share information. The applications that arise from this initial idea would represent a revolution in the industry (smart homes that make autonomous decisions for the benefit of the user, intelligent transport systems that guarantee the availability and efficiency of the transport network, health care that automates the taking of personalized data, among many other advances which will improve the quality of life [1]). Although it seems that there is still a long way to go to reach these developments, advances in 5G cellular network technology are noteworthy [2]. However, this progress towards a world in which everything is interconnected carries several risks since all devices share information that can be highly sensitive, and only one security breach in any of the communications would allow the interception or alteration of data. In addition, the security of transmissions with very restricted resources must be ensured, given the enormous number of devices connected and the speed with which they have to send the information. The field that studies information protection methods is called *Cryptography*. In essence, Cryptography is responsible for altering the messages sent so that, if the message is intercepted, unauthorized

* Corresponding author.
E-mail address: javiergv@fdi.ucm.es (L.J. García Villalba).

https://doi.org/10.1016/j.amc.2022.127116

recipients cannot obtain information from it. To do this, the message must be encrypted and after being sent, the receiver of the message must decipher it by using a key (shared exclusively by the sender and receiver). One of the most helpful mechanisms for this purpose is the generation of random numbers [3]. The need for such sequences is evident in a number of areas such as, for example, in Cryptography [4–7], Security [8–10], Simulation and generation [11–16] among others. If the data are encrypted with a set of sequences of random numbers, it is practically impossible for the sent message to be deciphered or altered since the same decryption methods would not work for each sequence (each one is different because they are random-generated).

Obtaining random sequences is a complex task since computers are deterministic machines: they need a set of predefined instructions for the generator to work. Therefore, the generated streams are not random, there is a pattern behind them that can be detected. An ideal option [17] would be to use physical phenomena such as thermal noise or radioactive decay to give true randomness to the sequences since these events are inherently unpredictable. However, obtaining data from these phenomena has a high computational cost, so they are not very useful for real devices given the characteristics of IoT systems. In addition, as they are completely random, there is no key with which the transmitted information can be deciphered, so not even the receiver could know the message. Therefore, what is desirable is to create generators that are fast and computationally simple, but also capable of producing random sequences. To compare existing generators statistical tests are used. These tests calculate a value relative to a specific characteristic of the sequence (the number of zeros and ones it has, the arithmetic mean, the number of times that more than two zeros or consecutive ones occur, etc.), and obtain the probability of getting this value if the studied sequence were random. If many generated streams fail a test there is a vulnerability in the generator, so it may be changed or re-implemented. It should be noted that this is not true in reverse: if a generator passes all tests, it does not mean that it is random since the right test matching the hidden pattern of the generated sequences may not have been found.

In general, statistical tests are grouped into sets called batteries or suites. Batteries try to take the best tests available, seeking a balance between the effectiveness of the tests and how long it brings to analyze each sequence. Essentially, each of the tests tries to look for different vulnerabilities in our studied flows. However, it may be possible for two tests to measure similar characteristics, although it may not appear so at first instance. Therefore, a generator that fails one of the tests will also fail the other with high probability, so it would be considered as bad as another one that fails two tests that measure different properties (when actually it is not). In addition, having two tests that can be considered redundant increases the computational execution time unnecessarily.

Independence analysis is responsible for measuring the relationship between statistical tests. Traditionally, the study of interrelationships has been carried out using Pearson's correlation which measures the degree of linear relationship that exists between the variables. Some interesting references that consider this approach are for example [18,19] or [20] among others. However it is not capable of detecting non-linear dependence relationships, which is why Mutual Information measure has recently been proposed (see [21]). Mutual Information is a quantity that measures the information that can be derived about one variable from another. In [21] this measure has been used for the first time to examine the NIST SP 800-22 battery. The results showed that Mutual Information is a tool that can provide novel advances in the study of the independence of hypothesis tests in a battery.

The main objective of this paper is to verify the effectiveness of Mutual Information applied to the study of independence in randomness tests. To do this, the measure has been applied to different tests batteries and the results have been compared with those obtained with the traditional approach using Pearson's correlation. That opens a new objective: to search for new vulnerabilities corresponding to the dependence on the batteries studied, which may lead to their improvement (discarding redundant tests or modifying the implementation of tests) and, in the future, contribute to the development of one "single battery" that includes the best (and independent from each other) statistical tests.

This paper is organized as follows: in Section II the necessary definitions of the measures are explained, in Section III the description of the experimentation and the analysis of three well known batteries (DieHarder, TufTest and SmallCrush batteries) are shown. In Section IV some guidelines about the construction of an efficient battery are given. Finally, in Section V the principal conclusions of this research are given.

2. Key concepts

As mentioned above, the method traditionally used to analyze the dependencies between the tests in the batteries has been the Pearson's sample correlation coefficient ([22–24] among others). Given two samples $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_n\}$, it is defined the sample correlation coefficient $r_{X,Y}$ as:

$$r_{X,Y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where $\bar{x}$ and $\bar{y}$ are the arithmetic means of X and Y respectively. The main disadvantage of this measure, as is already known, is that it is exclusively linear, it is not capable of detecting other types of dependencies. To overcome this drawback, the Mutual Information measure has recently been proposed to measure non-linear relationships between variables. Given

two random variables X and Y with values in S_x and S_y, the Mutual Information between them is defined as:

$$I(X,Y) = \int_{S_x}\int_{S_y} p(x,y)\, log\left(\frac{p(x,y)}{p(x)p(y)}\right) dxdy$$

where $p(x,y)$ is the joint probability function of X and Y, and $p(x)$ and $p(y)$ are the marginal probability distribution functions of X and Y, respectively. And, in its discrete form:

$$I(X,Y) = \sum_{i=1}^{n}\sum_{j=1}^{m} p(x_i,y_j)\, log\left(\frac{p(x_i,y_j)}{p(x_i)p(y_j)}\right)$$

Mutual Information between X and Y measures the reduction of the entropy for each variable, if the value of the other variable is known. It can also be expressed as [25,26]:

$$I(X,Y) = H(Y) - H(Y|X) = H(X) + H(Y) - H(X,Y)$$

where $H(X)$ and $H(Y)$ are the Entropy of X and Y respectively, $H(Y|X)$ the conditional Entropy of Y given X and $H(X,Y)$ the joint entropy of both variables. For more details about Mutual Information see for example [27] or [28].

3. Experimentation

In this section DieHarder, TufTest and SmallCrush (from Test U01) batteries are analyzed by classic correlation methodology and the algorithm described in [21] for the Mutual Information approach. For both measures 100 different tests for each pair of tests (both Pearson's correlation and mutual information) have been run, and a Kolmogorov-Smirnov (K-S) test on each set of 100 p-values have been applied, this provides a more reliable measure of their uniformity. The significance value α will be, in both measures, 0.001.

Two machines have been used for the analysis: (i) a Windows machine with Operating System: Windows 10 (64-bit), CPU: AMD Ryzen 3700XT (3.6 Ghz, 8 cores, 16 threads) and RAM: 64 GB and (ii) a Linux machine with Operating System: Debian (64-bit), CPU: Intel Core 6200U (2.3 Ghz, 2 cores, 4 threads) and RAM: 8 GB.

The implementation has been designed in Python and the code is available in https://github.com/mbrianlc/TFG/IndependenceAnalysis.

3.1. Analysis of dieharder test battery

The DieHarder test suite, created by Robert G. Brown [29], is an extension of the well-known DieHard battery [30], which modifies and improves the tests. In addition, more randomness tests have been added, both from other batteries such as NIST STS 800-22 [3] and own tests (such as RGB by Robert G. Brown). Its objective is to become the "Swiss army knife of random number test suites", the most powerful battery, so the author has designed it intending to include more tests in the future. It also includes some of the main random number generators to perform experiments without using external generators.

The test included in the battery are (some of them have the label "modified", which means that they have received improvements in DieHarder compared to the original version): Diehard Birthdays Test (modified), Diehard Overlapping 5-Permutations Test, Diehard 32 x 32 Binary Rank Test, Diehard 6 x 8 Binary Rank Test, Diehard Bitstream Test, Diehard Overlapping Pairs Sparse Occupance (OPSO) Test, Diehard Overlapping Quadruples Sparce Occupancy (OQSO) Test, Diehard DNA Test, Diehard Count the 1s Test (stream) (modified), Diehard Count the 1s Test (byte) (modified), Diehard Parking Lot Test (modified), Diehard Minimum Distance (2d Circle) Test, Diehard Minimum Distance (3dSphere) Test, Diehard Squeeze Test, Diehard Sums Test, Diehard Runs Test, Diehard Craps Test, Marsaglia and Tsang GCD Test, STS Monobit Test, STS Runs Test, STS Serial Test, RGB Bit Distribution Test, RGB Minimum Distance Test, RGB Permutations Test, RGB Lagged Sums Test and RGB Kolmogorov-Smirnov Test. In general, the tests are applied on 32-bit integer sequences.

There are two ways to access the battery: (i) The original version in the form of a.tgz file, which can be found on the official page.[1] It can also be installed and accessed from Linux without manually downloading the file (dieharder command) and (ii) a version for R language, called RDieHarder, which has much of the functionality of the original battery, and incorporates some new tools. However, both versions have drawbacks:

1. When this battery is executed on a set of n sequences, it calculates n p-values for each test, on which it performs a K-S test. As independence is wanted to be studied, only the list of p-values is needed, however, in the original version only the result of the K-S test is provided. On the other hand, this is possible in the RDieHarder version (through the command dh$pvalues, assuming that dh stores the results of a test).
2. RDieHarder cannot run all the tests at the same time. Fortunately, this is not an impediment to carry out the tests on identical sequences since most of the available generators can be initialized with a seed (we take the same one for each test) or use an external file as a generator.

[1] https://webhome.phy.duke.edu/~rgb/General/dieharder.php

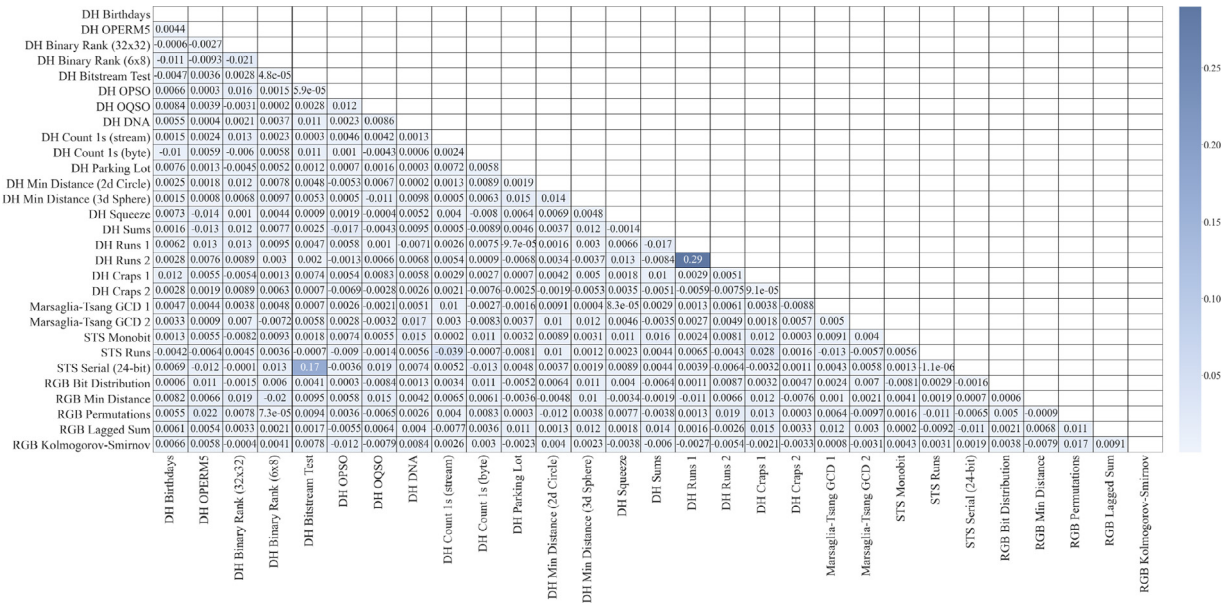

Fig. 1. Pearson's correlation (DieHarder): results.

3. The size of the sequence on which each test is applied is fixed and, in general, different in each case (with very varied sizes, from 796 bytes to 80 MB), so results between several tests cannot be compared. The original version allows adjusting the size of the sequence to which each test is applied, although the author warn that some tests are not prepared to be executed in different sizes, so they require a previous analysis to decide if they can be modified. At RDieHarder, these sizes cannot be different in any case. We propose the following solution:
   (a) Take sequences of approximately 80 MB, which are the size of those used by the GCD test.
   (b) For each test T_i, which uses sequences of k MB, consider the integer m closest to $\frac{80}{k}$, so if we join m sequences of size k we get close to 80 MB.
   (c) Execute for the T_i test m sequences of size k, obtaining k p-values, on which we carry out a K-S test. The result of the test will be the returned p-value. That is, in essence, what the battery does, so no additional implementation is needed.

   With this solution, the sequences are still not the same size in many cases. Ideally, it would be best to use the least common multiple of all possible sizes as the final size, but the result is so large that it is unfeasible (approximately $3.19 \cdot 10^{18}$ integers $= 1.2 \cdot 10^{47} TB$ per p-value). It is not the best solution in our opinion since we are not performing the original tests (it is as we divide the sequence into m "blocks" on which we perform the test, using the K-S test as the representation of the m obtained p-values), but it can be used as a first analysis of the battery (of which, up to now, there are no independence studies). Since we will use the result of the K-S tests as the p-values, we will stick with the original version of DieHarder, as it is faster and has more functionality than RDieHarder. It is left, as future development, to carry out a more in-depth analysis on each test, checking for which sequence sizes it maintains its efficiency. Thus, we could determine an ideal sequence size to use with all the tests and have more reliable results.

10^4 sequences of size 84 MB will be analyzed. The used generators will be: dev/urandom [31], CryptGenRandom()[32], Python secrets() and qRNG.

There are 4 tests that return more than one p-value: Diehard Runs, Diehard Craps, GCD, with two p-values, and STS Serial, with 24 p-values. For the first three we will keep both versions: although it is quite likely that we will find dependencies in the DieHard Runs test (the number of increasing and decreasing runs should be similar), we cannot justify that there is a relationship between the variants of the other two tests (For example, for the Craps test, the number of wins can be independent of the number of steps to reach it, and the two p-values are obtained in different ways). Regarding the STS Serial test, the author recommends using its version of larger blocks (24 bits) since it encompasses the rest (if a block of 2 bits appears more times than usual, the 24-bit blocks that include this one will also appear more times than average). For informational purposes, the results related to the Diehard Sums test will be included, although they should not be taken into account as the use of the test is not recommended.

In Fig. 1 correlations between obtained p-values are represented. The highest correlation observed is between both versions of DieHard Runs (0.287124), which, as expected, indicates that there were certain similarities between the two variants. However, this does not happen with DieHard Craps or GCD, with very low correlations (-0.000091 and 0.005006, respectively), so it seems that here the test variants are differentiated. The next notable correlation is between STS Serial

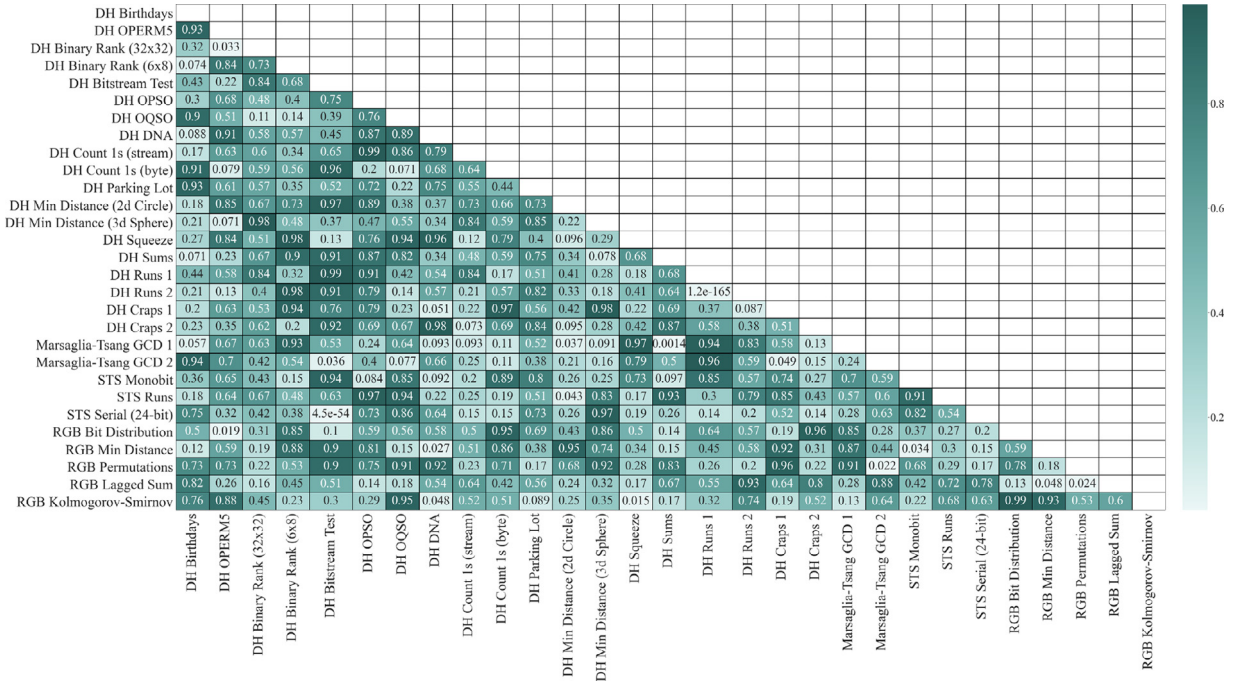

Fig. 2. Pearson's correlation (DieHarder): K-S.

and DieHard Bitstream (0.170163). That may make sense since STS Serial counts 24-bit patterns, while DieHard Bitstream counts 20-bit patterns that don't appear. Of the rest of the pairs, only two stand out slightly: STS Runs with DieHard Count the 1s (stream) and with DieHard Craps 1. Significance will indicate whether the values are high enough to be considered dependencies.

K-S test is only failed by the two high correlated pairs (see Fig. 2): DiehHard Runs 1 and 2 ($1.2 \cdot 10^{-165}$) and STS Serial with DieHard Bitstream ($4.5 \cdot 10^{-54}$). The rest of the pairs passed the test, including the two that had slightly more correlation than the rest (0.249838 between STS Runs and DieHard Count the 1s (stream), and 0.851792 between STS Runs and DieHard Craps 1), which shows us that this difference was not significant. These results also show that there are no dependencies (at least linear) between the two versions of Craps and GCD, which also pass the test.

Results related to Mutual Information measure are represented in Figs. 3 and 4. As it can be seen, Mutual Information matrix is not able to find more dependencies. Only the two pairs already discussed have a higher value than expected: 0.021129 between DieHard Runs 1 and 2 and 0.007491 between STS Serial and DieHard Bitstream. It can be seen a high reduction of the values concerning the Pearson's correlation (13.59 and 22.72 times lower, respectively), which reinforces the idea of (essentially) linear dependence, especially in the second case. In the absence of knowing the significance, none of the other pairs have a high correlation. The significance results confirm the preceding correlations. The two usual pairs fail the K-S test ($1.5 \cdot 10^{-89}$ between DiehHard Runs 1 and 2 and $4.1 \cdot 10^{-17}$ between STS Serial and DieHard Bitstream). Meanwhile, the rest of the pairs remain in a normal range (K-S test results between 0.019 and 0.98). That means that, for this battery, no new dependencies have been found apart from the linear ones.

Fig. 5 shows the distribution of the p-values. For this battery, the distributions are not very helpful: all pairs have an (apparently) uniform distribution in $[0, 1] \times [0, 1]$, even the two that we know have dependencies. It can be expected since the correlations are not very high, although a slight difference between the versions of STS Runs is assumed.

In principle, it is the battery that has given the best results because even being the most extensive (with variations, 29 tests in total), it is the one that has shown the least amount of dependencies (only two pairs). DieHard Runs tests have the highest correlation due to the similarity between the number of ascending and descending runs. To calculate all the runs of a block you need to go through it once (linear in m, with m the block size). Therefore, whether we want the increasing and decreasing runs or if we only want a single type of runs, we must go through all the blocks of the sequence. The p-value is obtained for both tests with a Chi-square test on 10 elements ($O(1)$), so calculating this goodness-of-fit test once or twice is not decisive in complexity. Thus, despite having dependencies, deleting a test would hardly affect the execution time, so we keep both. As for the STS Serial and DieHard Bitstream tests, we suggest keeping both tests as well. Remember that the dependency has been obtained with a specific version of STS Serial, so there could be another that shows less correlation. It remains as future development to repeat the experiments with different variants of STS Serial to find the best version.

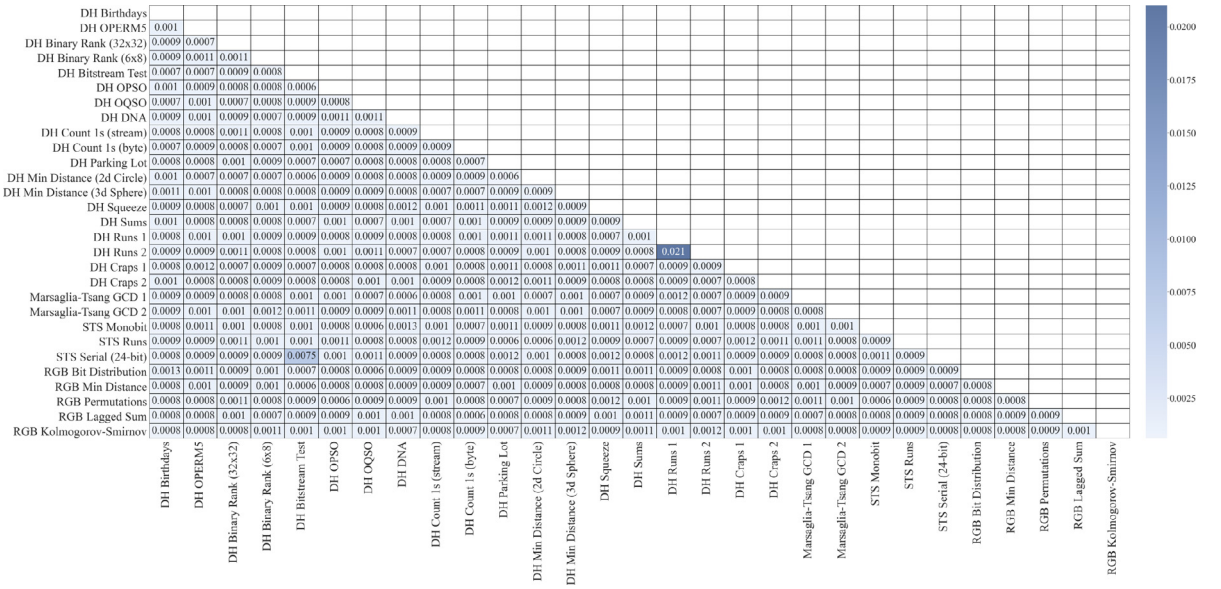

Fig. 3. Mutual information (DieHarder): results.

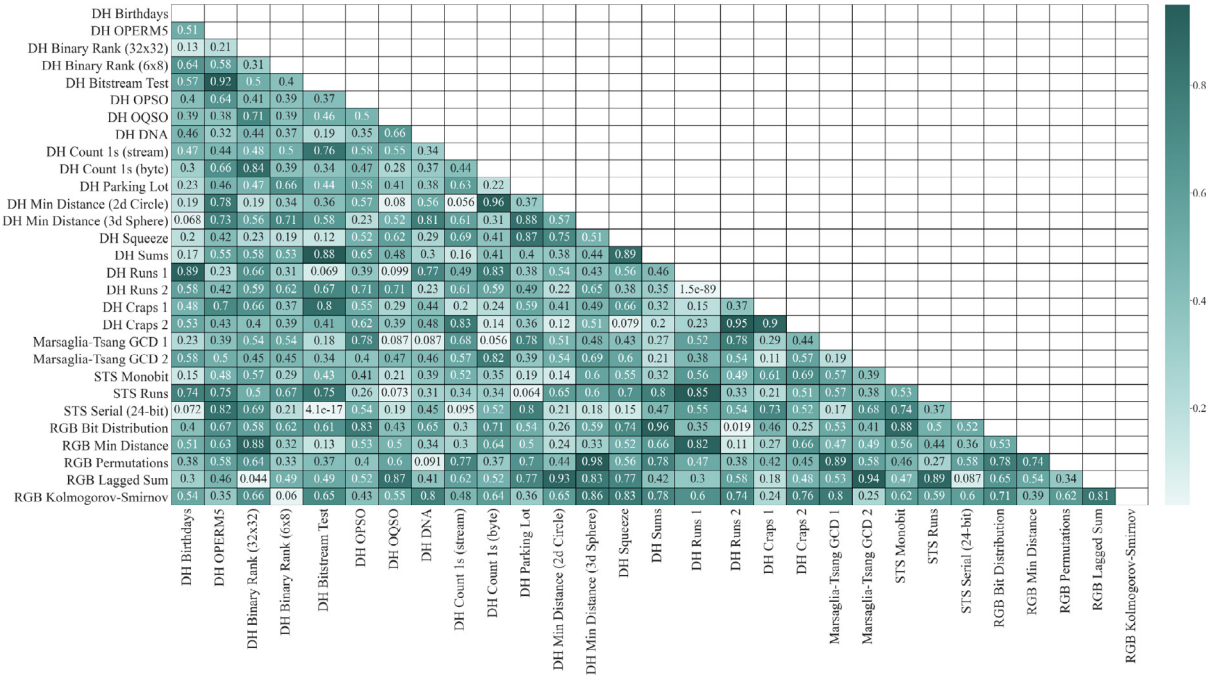

Fig. 4. Mutual information (DieHarder): K-S.

3.2. Analysis of tuftest battery

Many random number generators, especially congruential generators, cannot pass the three tests, created by George Marsaglia and Wai Wan Tsang: GCD test, Gorilla test and Birthdays Spacings test, whose descriptions can be seen in [33]. According to their authors, we can see these tests as a "distilled version" of the DieHard battery: they are fewer, but maintain DieHard's essence, and they are easier to apply. As the Diehard battery, the tests are applied on 32-bit integer sequences, and the sequence's size depends on the test.

The authors provide the file "tuftests.c" in which the implementation of each test comes. However, the tests are applied to different sequences, so in principle, we cannot compare their p-values. Two problems arises:

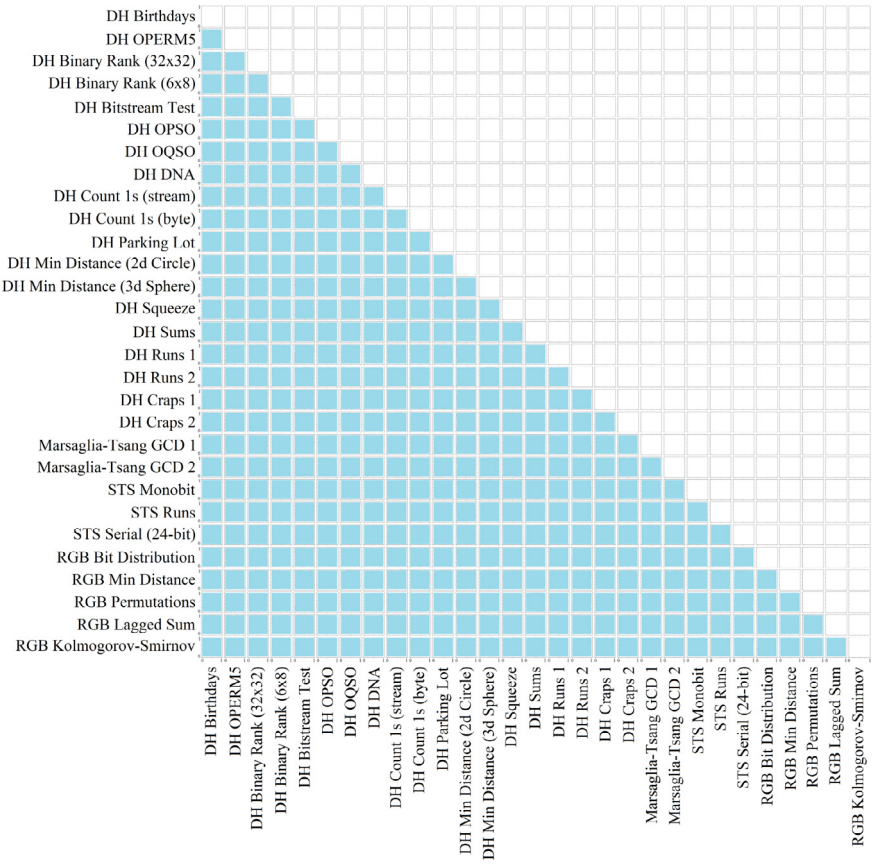

Fig. 5. Dispersion matrix.

- When a test is changed, the implementation does not restart the generator. This problem is easy to solve: specifically, we use fixed-size files, and every time we run a test, we specified that it has to be read again from byte 0. The file also resets for each p-value of the Gorilla test since originally the test on each bit was on different sequences. On the other hand, it has not been necessary to do it in the two p-values of GCD since the test obtains them from the same streams of numbers.
- Each of these tests is done on sequences of different sizes: (i) GCD: $10,000,000 \cdot 2 = 20,000,000$ integers $= 80,000,000$ bytes $= 80$ MB (it may need more integers as it discard null values), (ii) Gorilla: $2^{26} + 25$ integers $= 268,435,556$ bytes $\simeq 268$ MB and (iii) Birthdays Spacings: $5000 \cdot 4096 = 20,480,000$ integers $= 81,920,000$ bytes $\simeq 82$ MB. Two options can be considered:
  - Modify the tests so that they are the same size (or very similar). In particular: (i) increase the value of n $(10,000,000 \rightarrow 33,554,444)$ of GCD: $33,554,444 \cdot 2 = 67,108,888$ integers $= 268,435,552$ bytes and (ii) increase the number of executions $(5000 \rightarrow 16,384)$ of Birthdays Spacings: $16,384 \cdot 4096 = 67,108,864$ integers $= 268,435,456$ bytes. In both cases the size will be ~ 268 MB. That, for the two tests, would only translate into having more values in the final Chi-square test, so if we make the necessary adjustments it should not imply a drastic change in the test (although, in any case, they would no longer be the original test).
  - The GCD and Birthdays tests apply to approximately $\frac{1}{3}$ of the total data. Therefore, for these tests, we can divide each sequence into three parts of the same size, obtain three p-values and perform a K-S test on them. The p-value that they would return would be, in that case, the result of the K-S test. It is the same solution we used in the DieHarder battery which, as it was said previously it is not our favorite because we are not executing the original tests.

  Finally, we will carry out the first option because we consider that it is more similar to the initial battery. This also means that the GCD test is now different from the one used in DieHarder, so the results corresponding to these tests may be different. As the GCD test may need more integers, we will take files of size 270 MB so that the tests can be done without problems.

10^4 sequences of size 80 MB will be generated. Due to the high complexity of the tests (especially the Gorilla test), only dev/urandom has been used in the experimentation. As in our previous experimentation, we run 100 experiments in order to study the significance with K-S test with level $\alpha = 0.001$. For the GCD test both versions will be considered, denoted

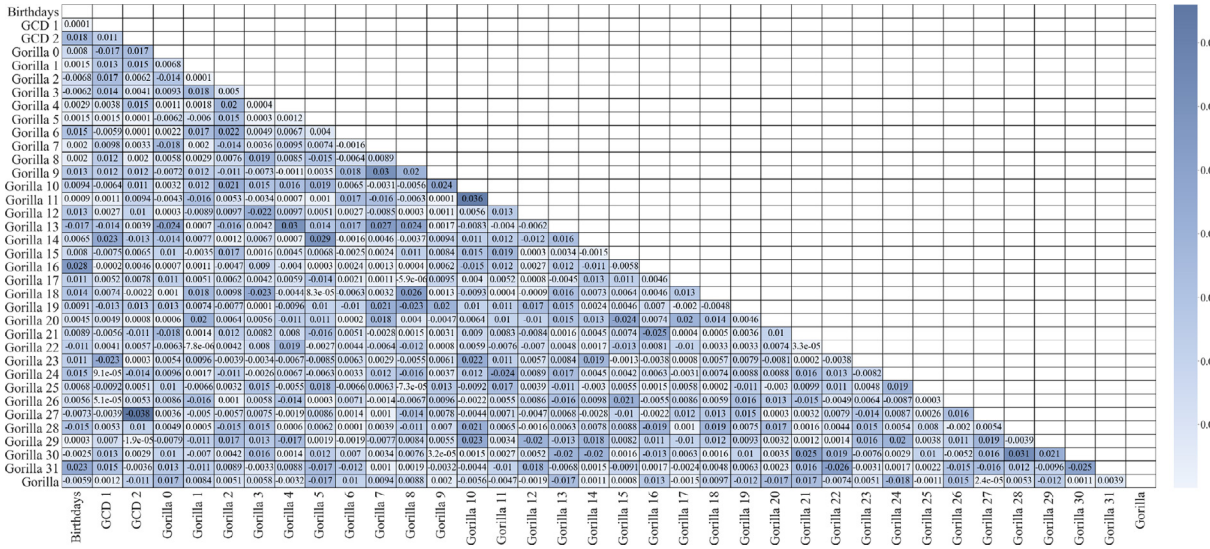

Fig. 6. Pearson's correlation (TufTests): results.

respectively as GCD 1 and GCD 2. As for the Gorilla test the results applied to each bitfor informational purposes will be shown. These results should not be taken into account since, by definition, they are the same test, but applied to different sequences (all 0 bits, all 1 bits, etc.). Therefore, if there is no dependency between bits in our generator, the Gorilla tests on each bit should be independent of each other. However, they can be used to compare them with the results of other pairs, thus giving an idea of their independence.

The results of our experimentation are quite positive, as no dependencies have been found. In Fig. 6 the results corresponding to the correlation analysis are shown. The correlations are low, with $m = -0.037214$, obtained between the Gorilla test for bit 27 and GCD 2, as the maximum (in absolute value). The rest of the results remain in this line, although there are some correlations somewhat lower than the rest, around 10^{-5}.

The results of the K-S test (see Fig. 7) clear up any doubts that could exist in the previous matrix. All the pairs passed the test, with the value of the Gorilla tests of 29 and 14 bits (0.014102), which we already knew should be independent, the one closest to failing the test. With respect to the analysis of Mutual Information, this measure is also not able to find dependencies as can be seen in Fig. 8. Again, no remarkable results are produced, with 0.001432 as the maximum correlation (which also occurs between the 1-bit and 23-bit versions of Gorilla, in principle independent). In addition, the range of values is smaller than before (the difference between the maximum and the minimum, obtained again between Gorilla variants, is 0.000829). Removing the Gorilla versions, the rest of the correlations are less than 0.001102 (between GCD 2 and Birthdays), which again suggests that the tests are independent.

Significance remains in line with expectations (see Fig. 9): the p-values pass the K-S test (with a minimum value of 0.004621, enough to pass the test, and also achieved again between two Gorilla tests: Gorilla 22 and Gorilla 10).

In Fig. 10 is represented the dispersion matrix, as it can be seen, all the pairs present a uniform distribution of their p-values in $[0, 1] \times [0, 1]$.

3.3. Analysis of smallcrush of TestU01

One of the most relevant battery is TestU01, created by Pierre L'Ecuyer and Richard Simard [34]. This library collects some of the best statistical tests of randomness studied and improves them, generalizing them or adding new steps. In addition, it features a wide variety of pseudo-random number generators, predefined batteries, and some extra tools (for graphs or analysis). All this allows great flexibility in the randomization study (for both tests and generators), unlike most batteries. TestU01 battery consists of numerous randomness tests from various sources. In addition, most of them have several versions, as they tend to depend on several parameters. Those tests are grouped on a wide variety of preset batteries (sometimes a battery has different versions of the same test). Most of these batteries have a high computational cost, for example, BigCrush has 31 tests (which, counting different versions, increases to 106), uses 2^{38} numbers of 32 bits (more than 1 TB) and around 8 hours of duration to obtain a single p-value on all tests. This high complexity is because these batteries were designed to rigorously measure our sequences and generally need few p-values to give a correct decision about a generator. However, this is a problem for our analysis: although the battery may not require many p-values for calculating the efficiency of the generators, it is needed a considerable number of them to conclude independence. Fortunately, a sub-battery in TestU01, called SmallCrush (10 tests, which provide 15 p-values) allows to make an initial analysis in less time. This battery was created to quickly rule out low-random generators, leaving the more complex batteries for more extensive

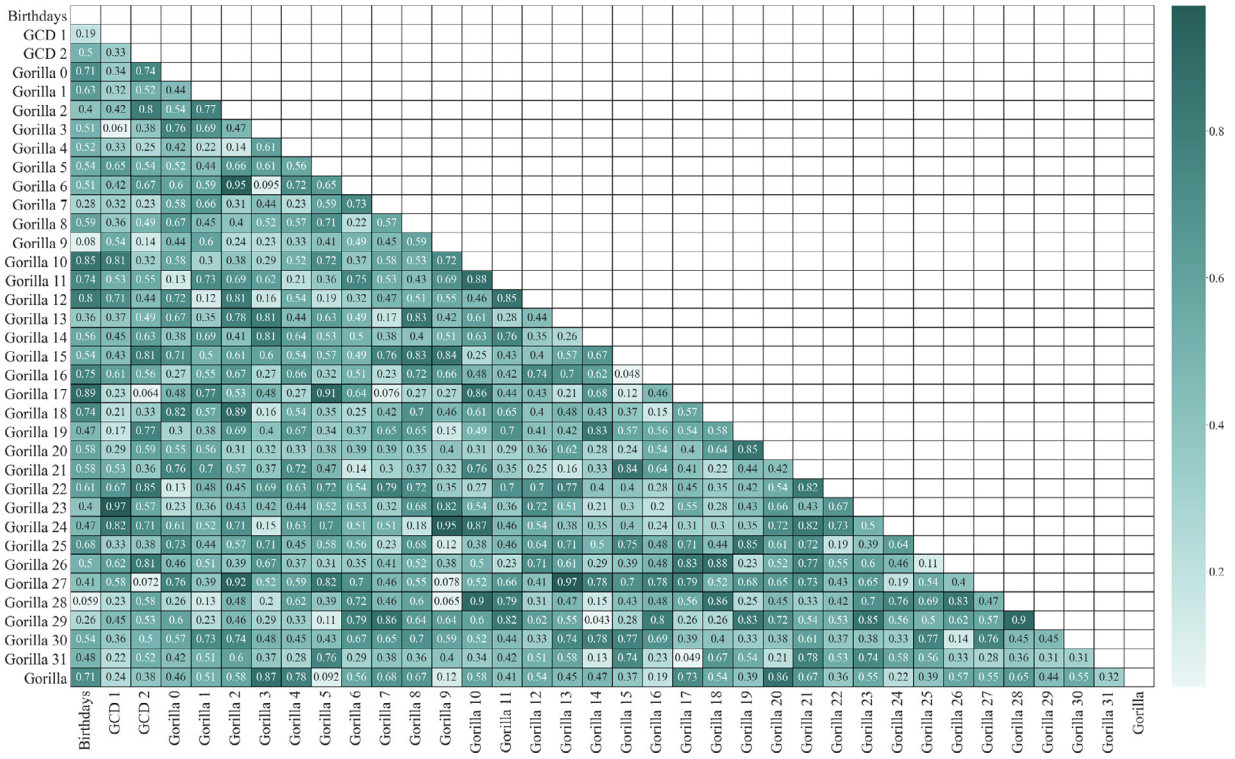

Fig. 7. Pearson's correlation (TufTests): K-S.

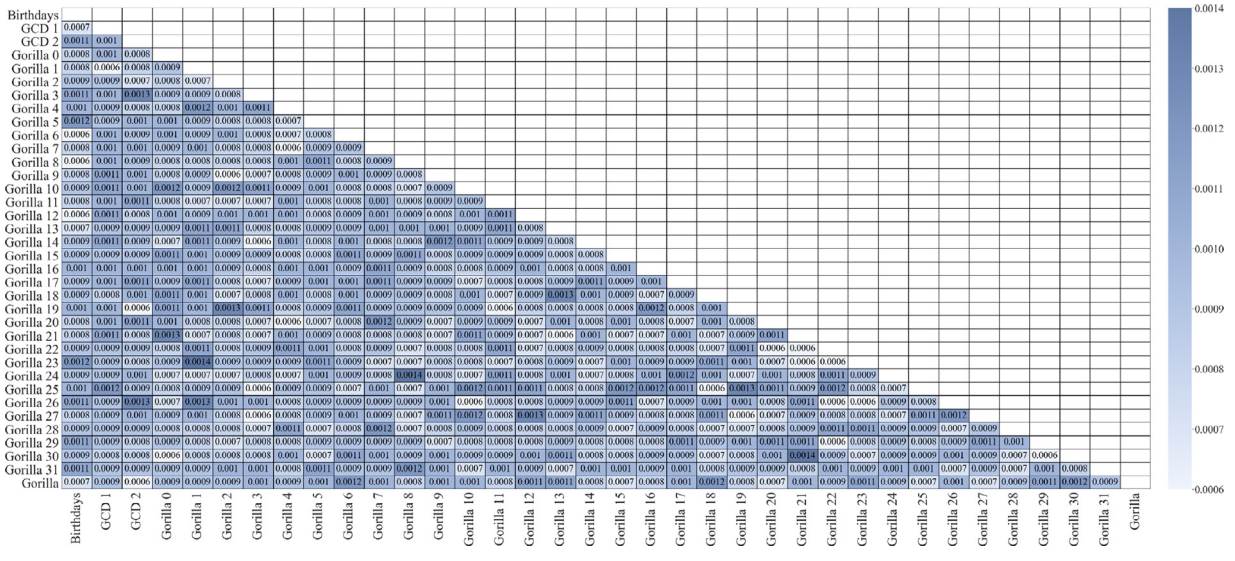

Fig. 8. Mutual information (TufTests): results.

testing. Even so, we would like to analyze the rest of the batteries in this module in the future, given its current relevance in this field.

Like the TufTests battery, this one operates on 32-bit blocks, but now each block is considered a real value in [0,1) (instead of an integer), using the transformation $N_i \rightarrow N_i' = \frac{N_i}{2^{32}}$. Some tests, however, use the original integer values or take the s most significant bits of each block.

Test included in SmallCrush are: smarsa_BirthdaySpacings, sknuth_Collision, sknuth_Gap, sknuth_CouponCollector, sknuth_SimpPoker, sknuth_MaxOft, svaria_WeightDistrib, smarsa_MatrixRank, sstring_HammingIndep and swalk_RandomWalk1 [34].

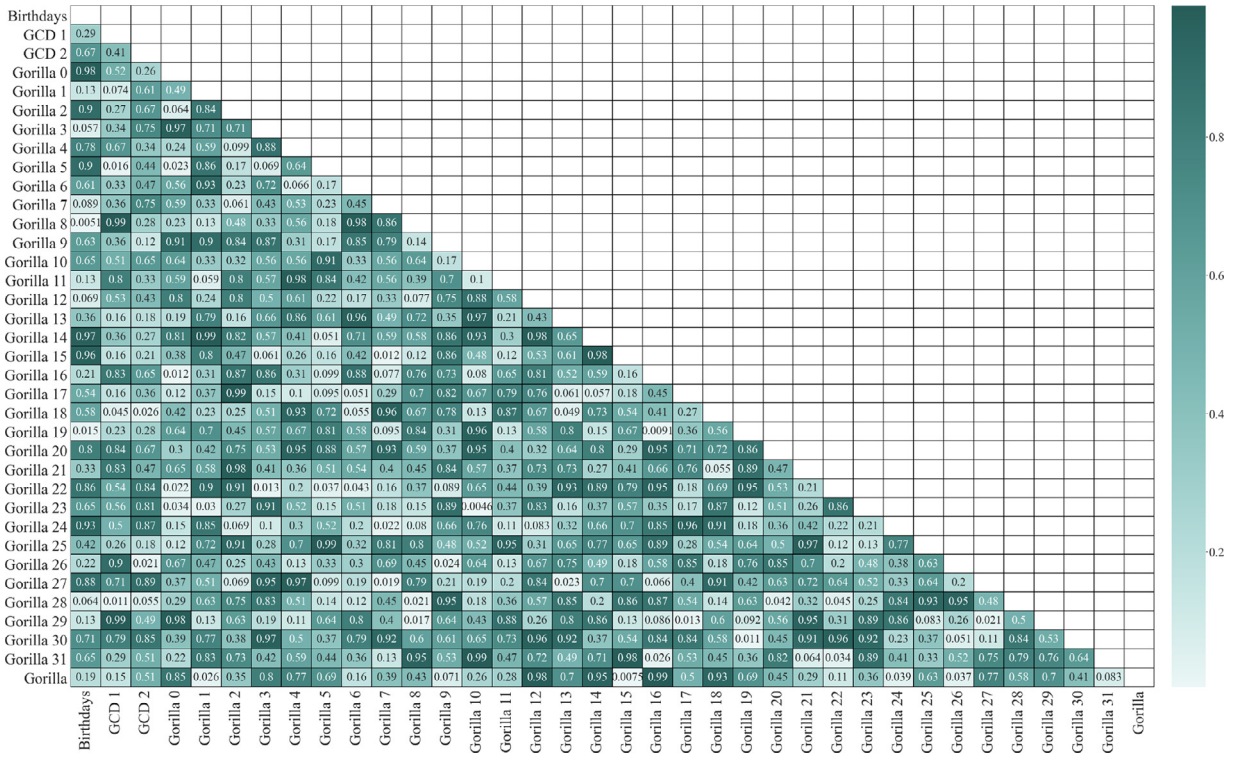

Fig. 9. Mutual information (TufTests): K-S.

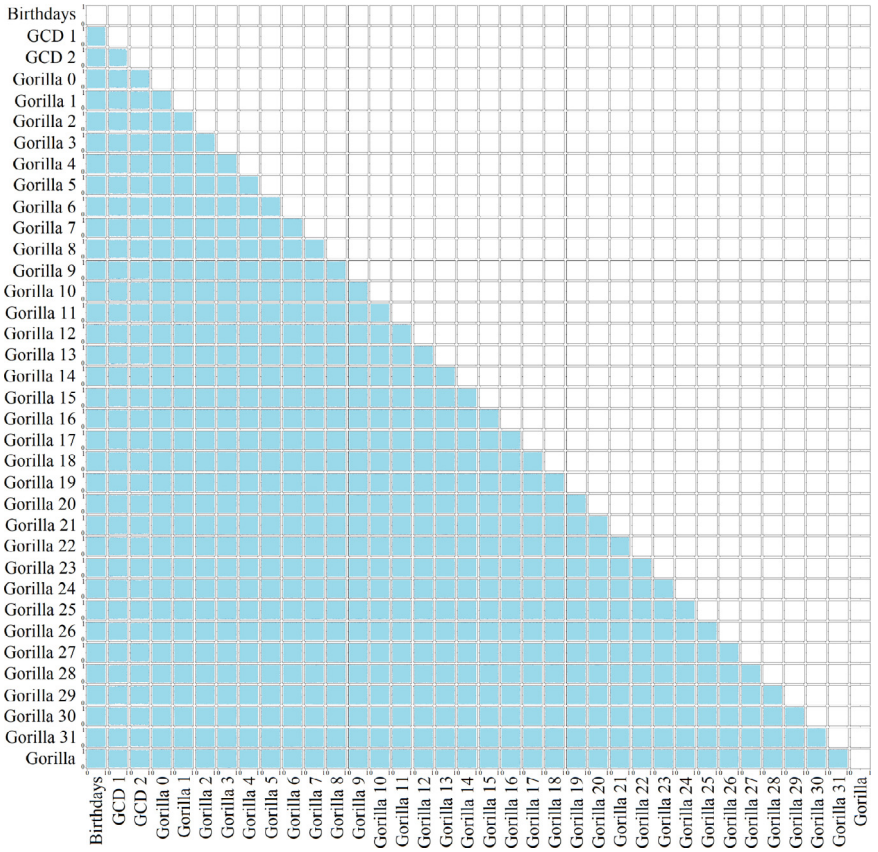

Fig. 10. Dispersion matrix.

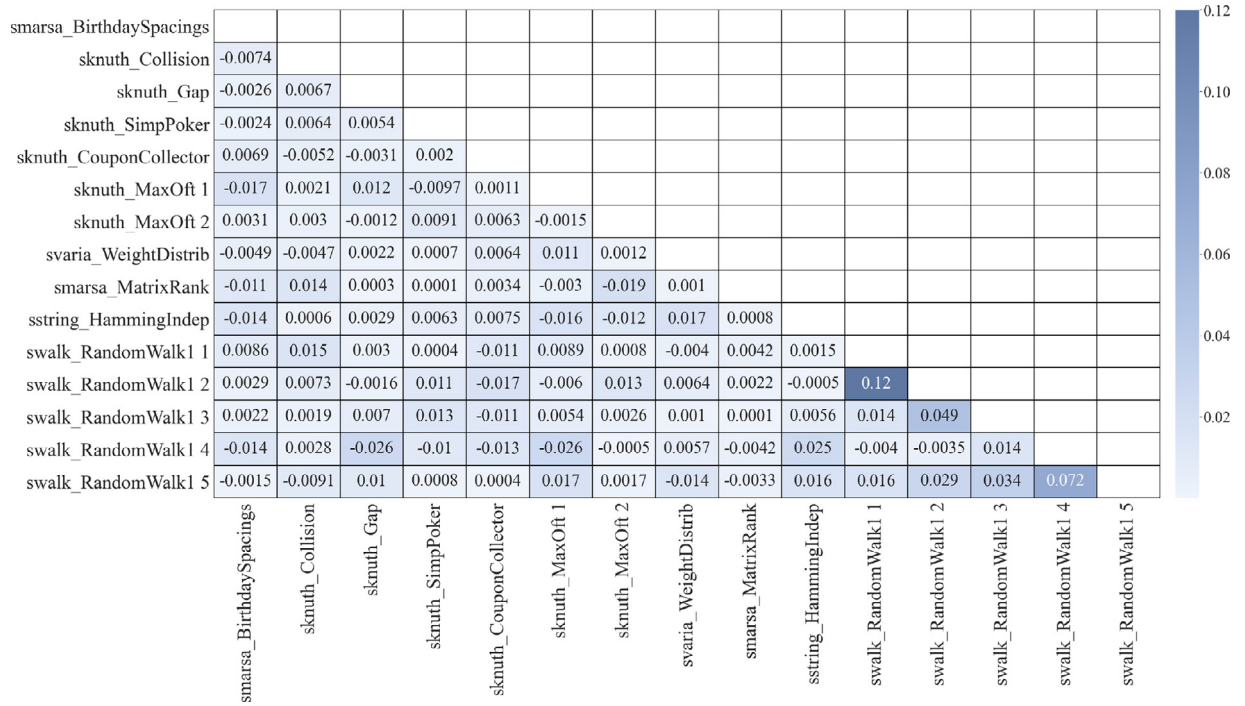

Fig. 11. Pearson's correlation (TestU01 - SmallCrush): results.

The authors provide an implementation of their batteries on their website.[2] In the experimentation bbattery_SmallCrushFile function has been used to analyze external files. To make it easier to use with some generators like dev/urandom, we have slightly modified the function so that it accepts binary files and transforms them into real numbers in [0,1) (something it already does in other batteries, but not in this one, where originally it only accepts real number files). An advantage of bbattery_SmallCrushFile is that it applies the same sequence to all tests, so we don't have to make any further modifications to the battery (as we did in TufTests). Again, the sequences are of fixed size: according to the user guide, a little less than 51,320,000 32-bit numbers = 205.28 MB are required, so that will be the size of our files.

10^4 sequences of 205.28 MB has been generated.For the significance analysis 100 experiments were performed and a K-S test of level $\alpha = 0.001$ was made. The used generators will be dev / urandom and Python.secrets() because although the complexity of the tests is somewhat lower than before, using sequences of that size is expensive (in the case of qRNG the generation of streams is slow, and in the case of CryptGenRandom() we have to transfer the data from Windows to Linux machine). As the results are similar for both generators, only those from dev/urandom will be shown.

In Fig. 11 results corresponding to the Pearson's correlation are shown. In general, the correlations are low. The most notable occurs among the p-values associated with tests 1 and 2 of swalk_RandomWalk1 (statistics H and M), with a value of 0.123912. The following highest correlation is not very remarkable: 0.071821, obtained again between two swalk_RandomWalk1 tests, specifically between 4 and 5 (R and C statistics). The rest of the correlations is less than 0.005. The results of the K-S test validate the results for the two pairs already described, see Fig. 12. The test returned a p-value of $2.4 \cdot 10^{-9}$ for tests 1 and 2 of swalk_RandomWalk1, and 0.000823 for tests 4 and 5 of swalk_RandomWalk1. A lower result than normal, although sufficient to pass the test, it is obtained between the swalk_RandomWalk1 5 and sknuth_Collision (0.009241) tests, which did not have a high correlation (-0.009102). We will have to see later results to draw more conclusions.

Regarding Mutual Information analysis Figs. 13 and 14 are shown. The two highest values are the same as before, although they are lower as usual: 0.002538 (48.8 times less) for tests 1 and 2 of swalk_RandomWalk1, and 0.001512 (47.5 times less) for tests 4 and 5 of swalk_RandomWalk1. As it was above-mentioned in previous batteries, that could be because these correlations are mainly linear. In addition, both correlations have been reduced by a similar factor. In addition to these two pairs, the Mutual Information finds another possible pair of dependent tests: swalk_RandomWalk1 1 (R statistic) and smarsa_BirthdaySpacings, with result 0.001448. All the other pairs, including the one between swalk_RandomWalk1 5 and sknuth_Collision, keep a low value. K-S test confirms the previous results, since the three commented pairs are the only ones that do not pass the test ($1.7 \cdot 10^{-5}$ between tests 1 and 2 of swalk_RandomWalk1, 0.000671 between tests 4 and 5 of swalk_RandomWalk1 and 0.000919 between swalk_RandomWalk1 1 and smarsa_BirthdaySpacings). There is also

[2] http://simul.iro.umontreal.ca/testu01/tu01.html

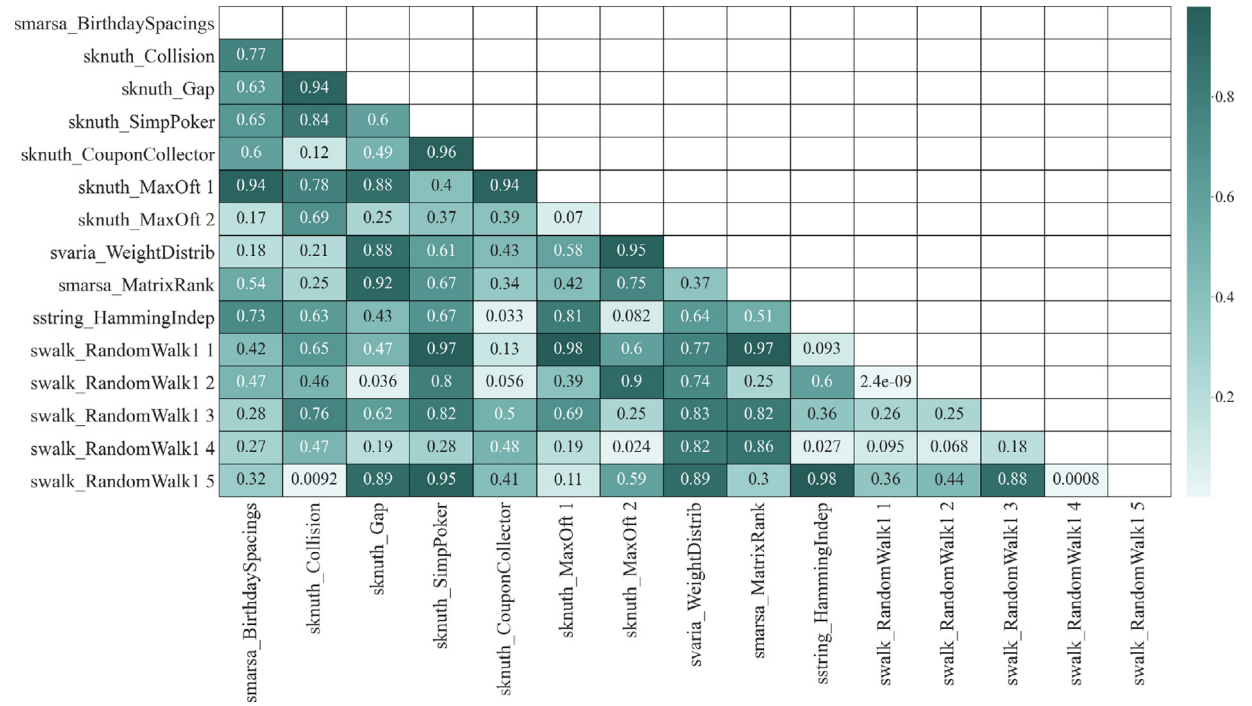

Fig. 12. Pearson's correlation (TestU01 - SmallCrush): K-S.

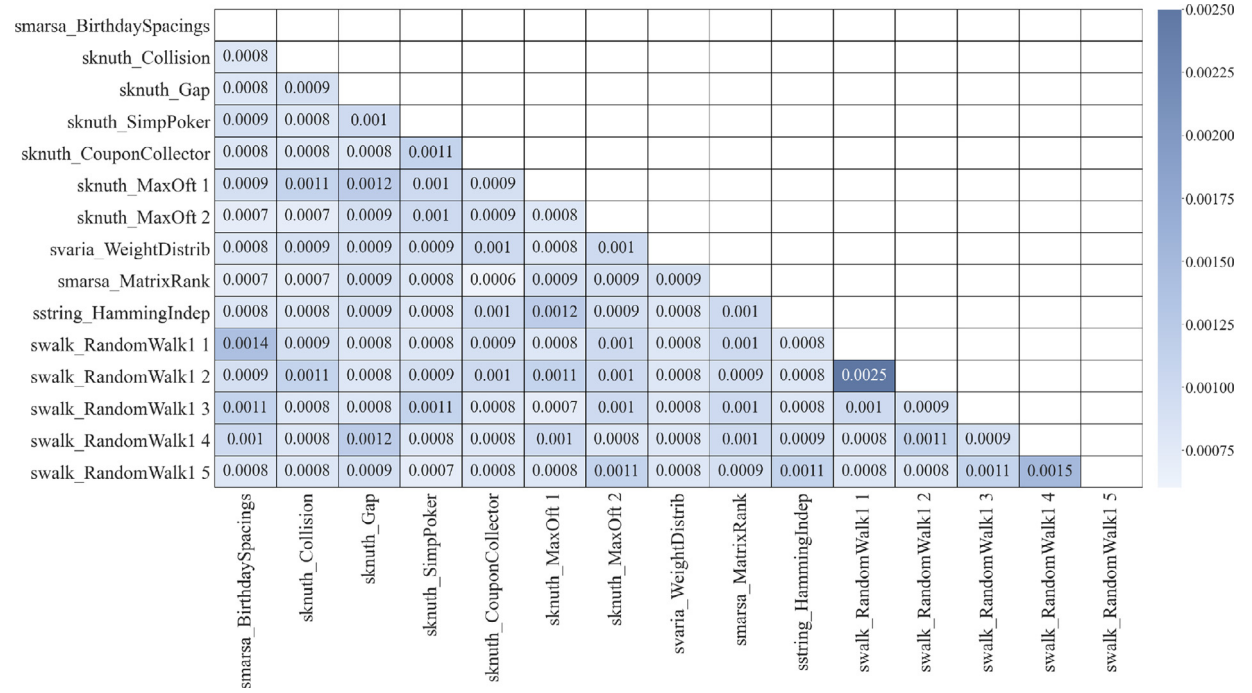

Fig. 13. Mutual Information (TestU01 - SmallCrush): results.

a pair about to fail the test: swalk_RandomWalk1 4 and sstring_HammingIndep (0.003488). Their Mutual Information was low (0.000917), but K-S test result invites us to make further experiments to ensure the independence of the couple. The one that does pass the test without a problem is swalk_RandomWalk1 5 and smultin_Multinomial (0.112701). Bearing in mind that all pairs of dependent tests with Pearson's correlation should be with mutual information, we can rule out our suspicions of dependence on this pair. Surprisingly, the two versions of MaxOft did not show any noticeable signs of depen-

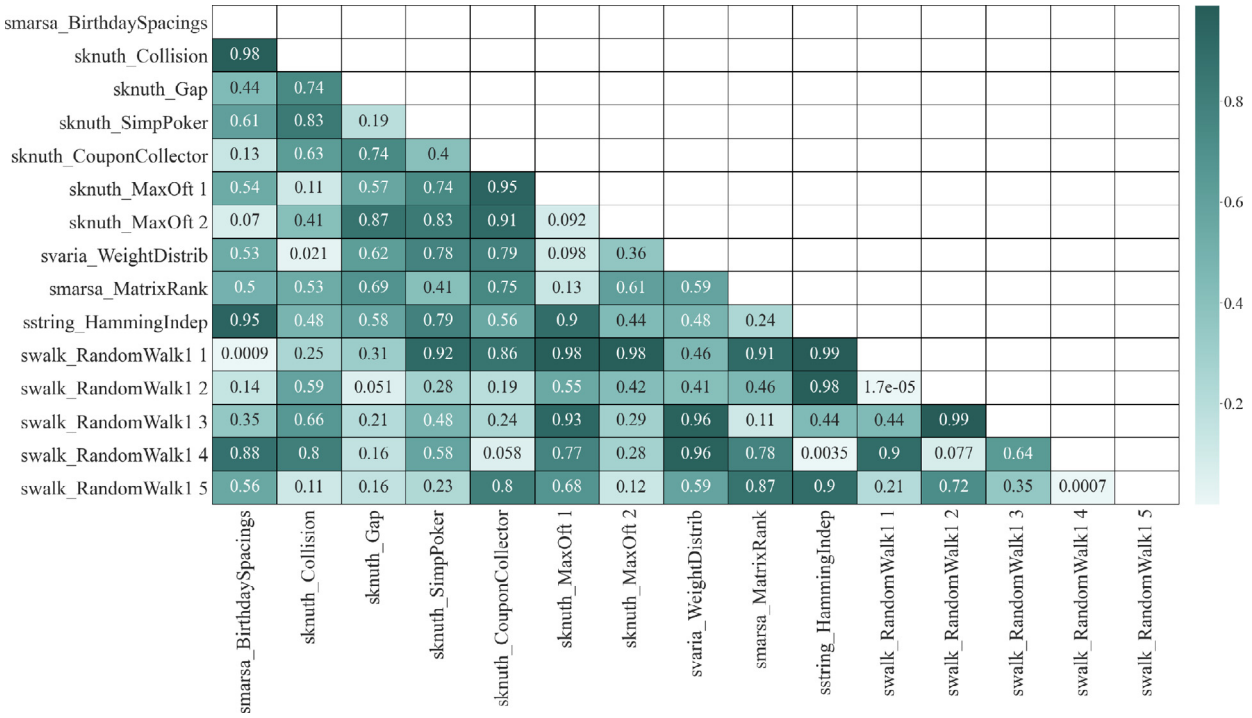

Fig. 14. Mutual information (TestU01 - SmallCrush): K-S.

dency. The results of the K-S test are not very high (0.071298 in Pearson and 0.091932 in mutual information), but they are, in principle, sufficient.

In Fig. 15 the distributions of the p-values is shown, all the pairs present a distribution of their p-values very close to uniform in $[0, 1] \times [0, 1]$.

After performing the tests, three significant dependencies have been found: two between two pairs from the same swalk_RandomWalk1 test (1 with 2, and 4 with 5), and one found with Mutual Information between swalk_RandomWalk1 5 and smultin_Multinomial. The most prominent occurs between swalk_RandomWalk1 1 and 2. To decide which of the two is better, the complexity of each test has been calculated. As the calculation of the p-value is the same (Chi-square test), the computation of the statistics have to be compared For the swalk_RandomWalk1 1 test, we obtain the statistic $H = \frac{l}{2} + \frac{S_l}{2}$, with l the size of the sequence and S_k the integer in which we find ourselves after k steps. That is, the calculation of S_l is linear in the size l ($O(l)$), so the test, which only adds 3 arithmetic operations, is also $O(l)$ ($l + 3$ operations). For the swalk_RandomWalk1 2 test, we obtain the statistic $M = \max\{S_k, 0 \leq k \leq l\}$. This is equivalent to calculating l steps and, in each one, comparing with the current maximum, so this test is also linear in l, but with $2 \cdot l$ operations (l steps and l comparisons). Both have the same complexity, but the first test has fewer operations, so if we had to discard a test it would be swalk_RandomWalk1 2.

As for the other two dependencies, test 5 of swalk_RandomWalk1 appears in both. The other two tests did not show, in principle, any relationship, so we would eliminate swalk_RandomWalk1 5.

As we said before, the content of TestU01 is extraordinary, and it gives us freedom both to modify tests and use new generators. With the SmallCrush battery, we have only covered a very small part of this library, and from here countless future work experiments are opened: testing with different variants of the tests, using other predefined batteries, repeating the tests with their generators, etc.

4. Creating the "definitive battery"

One of the main objectives in the field of cryptography is to arrive at a single battery of randomness tests, which is capable of encompassing the others. We have already seen some attempts, such as DieHarder or TestU01, that incorporate tests of different batteries, but there is still room for improvement. This "definitive battery" should meet the following requirements:

1. The tests must be effective and detect vulnerabilities in the least random generators.
2. All tests must be independent.
3. Its test should not have high complexity so that we can execute them in a reasonable time.

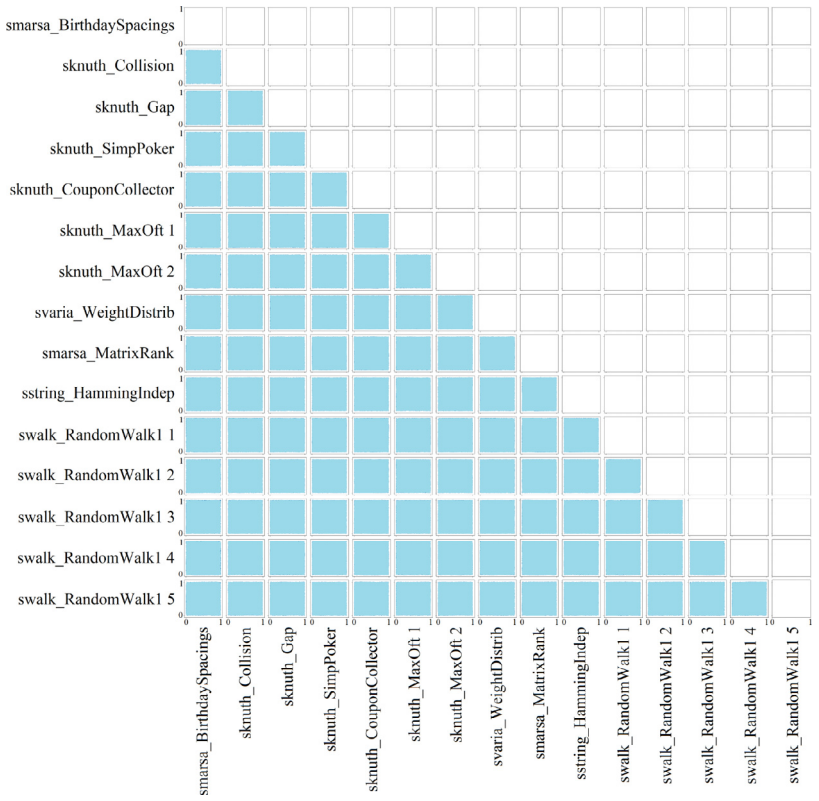

Fig. 15. Dispersion matrix for TestU01 - SmallCrush.

Ideally, we would start with the first condition, executing experiments with each test to measure its sensitivity and discard unhelpful tests, but that study is beyond the scope of this article, so we will assume that all tests are effective (in fact, when having worked with some of the best-known batteries, there have already been studies verifying the reliability or not of most of them).

The second requirement (independence) has been the main study of this article. After analyzing each battery, in addition to checking the usefulness of the mutual information measure, we have found numerous dependencies between tests (some already known, others novel). For each battery, we have seen which tests provide the least information about the generators analyzed and if perhaps it is worth more to discard them. Eliminating some tests would also reduce the battery run-time, something that influences the third requirement: efficiency. However, this section is much broader because even discarding dependencies we still have a considerable number of tests, many of them not very efficient. Although the timing of some batteries can easily be improved (such as FIPS 140-2, with a future implementation with p-values in C), some tests are slow or use a lot of data (such as TufTests). Should we include them even if they slow down the execution time of the battery? One possibility would be to do the same thing that is done in TestU01: have a small and fast battery to easily discard generators and use the authentic "definitive battery" (with all the tests), much more elaborate, but slow, for more exhaustive checks.

Although throughout the article we have found dependencies in most batteries and have discarded some tests, there is still a long way to go to reach the single battery: it is not enough to put together the independent tests of each battery, as we have to analyze the independence between tests of different batteries (which may have dependencies). Thus, we leave the development of this battery as future research. However, we propose a method to add tests to our battery that could be useful. Instead of analyzing mutual information on all the "valid" tests (quite an expensive process when dealing with many of them), it would be better to add the tests one at a time, so it is easier to discard tests. Let's look at each step:

1. For each of the batteries studied, choose a set of "valid" tests (effective, efficient, and independent of each other). That is what we have done (in the framework of independence).
2. Start from one of these "valid" test sets (we can take, for example, the ones from DieHarder, as it is the battery with the largest number of independent tests). These tests, $\{T_1, \ldots, T_n\}$ tests will already be part of the new battery.
3. Then, for each new test T, we add it to our set, and we calculate the mutual information between that test and each of the ones we had. If we find dependencies, we discard this test, otherwise, we add it to the battery.

5. Conclusions

After the analysis, it can be concluded that mutual information is a useful way to measure the independence between tests. This measure has been applied in this work to several batteries and it has been able to detect not only the correlations found by the Pearson's correlation but also new dependencies between tests.

Still, not all are advantages. The measure has several problems, the main one being its high complexity (for the significance), which prevents us from testing with larger sequence sizes or with batteries with more tests. It also has other disadvantages: if we restrict ourselves to linear dependencies, it is less informative than the Pearson's correlation since the latter informs if the relationship is positive or negative (being values between -1 and 1), something that mutual information does not do since it is always positive. In addition, it can find new types of dependencies, but it would be desirable to distinguish what they are. That would help the user to better understand the relation between two dependent tests, and use this information to modify (or discard) the tests.

For these aspects mutual information cannot be considered a "definitive measure" that eliminates other ways of detecting dependencies. We consider that it is more an additional measure that provides new information about the dependencies between tests, and that we can complement it with other measures. For example,wecanreducetheset of initialtestsonwhichto apply mutual information by first using Pearson's correlation to rule out tests with linear dependencies. For this project mutual information and the Pearson's correlation has been applied to all the tests of different batteries, but only to check if the mutual information was more beneficial. Now that we have demonstrated this, mutual information can be used on fewer tests and thus reduce execution times when analyzing new batteries. Furthermore, with this method, information about linear dependencies (since we use Pearson's correlation) is not lost, and we will know that the dependencies obtained with mutual information are not linear.

We consider that this discovery of Mutual Information as a new measure of independence represents a significant advance for the analysis of batteries of randomness tests, and it will probably be an important piece to build that ultimate battery that unifies the most relevant tests. In addition knowing the independence between tests can help us improve existing tests or develop new ones (seeking to make them independent), which will result in new pseudo-random generators that can pass all the tests and consequently guarantee more robust security measures, with all the applications that this entails. There are still many experiments to be done with this measure, and it will surely give us new and better results in the future.

As is well known, the study of randomness is a complex task, and each new progress offers new lines of development. Therefore, the arrival of this relatively recent (at least in the field of cryptography) and powerful tool opens up numerous avenues of advancement in the matter: (i) Given the efficiency of mutual information with some of the more popular batteries, the next step is to use it to test all possible batteries of randomness tests, (ii) Repeat the experiments with new sizes. Although we have not seen conclusive differences in the variations we have made, maybe they could occur in more extensive sequences. It would also be valuable to try new generators, (iii) As a complement to the previous point, analyze in-depth the tests of some of the batteries (such as DieHarder) to see if they can work with other sizes and if so, make the corresponding modifications, (iv) Follow the steps to find a "definitive battery" that guarantees the independence of their tests. Thus, we only have to focus on the efficiency and sensitivity of the battery, (v) Find new methods to detect specifically the type of dependency between tests with high correlations and (vi) Optimize the calculation of p-values in Mutual Information, or find a new way to measure the significance of the measurement, to extend the analyzes to more batteries or achieve more results with the ones we have studied.

Acknowledgments

This work has received funding from UCM Projects THEIA (FEI-EU-19-04), THEIA I (FEI-EU-21-01) and THEIA II (FEI-22-01).

References

[1] S. Kumar, P. Tiwari, M. Zymbler, Internet of things is a revolutionary approach for future technology enhancement: a review, J. Big Data 6 (1) (2019) 1–21.
[2] W. Ejaz, A. Anpalagan, M.A. Imran, M. Jo, M. Naeem, S.B. Qaisar, W. Wang, Internet of things (iot) in 5g wireless communications, IEEE Access 4 (2016) 10310–10314.
[3] L.E. Bassham, A.L. Rukhin, J. Soto, J.R. Nechvatal, M.E. Smid, E.B. Barker, S.D. Leigh, M. Levenson, M. Vangel, D.L. Banks, N.A. Heckert, J.F. Dray, S. Vo, SP 800-22 Rev. 1a. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, Technical Report, National Institute of Standards & Technology, Gaithersburg, MD, USA, 2010.
[4] R. Álvarez, J.J. Climent, L. Tortosa, A. Zamora, An efficient binary sequence generator with cryptographic applications, Appl. Math. Comput. 167 (1) (2005) 16–27 https://www.sciencedirect.com/science/article/pii/S0096300304004709, doi:10.1016/j.amc.2004.06.065.
[5] I. Baturone, M.A. Prada-Delgado, S. Eiroa, Improved generation of identifiers, secret keys, and random numbers from srams, IEEE Trans. Inf. Forensics Secur. 10 (2) (2015) 2653–2668, doi:10.1109/TIFS.2015.2471279.
[6] M.R. Khalili-Shoja, G.T. Amariucai, S. Wei, J. Deng, Secret common randomness from routing metadata in ad hoc networks, IEEE Trans. Inf. Forensics Secur. 11 (8) (2016) 1674–1684, doi:10.1109/TIFS.2016.2550424.
[7] D. Hurley-Smith, J. Hernández-Castro, Certifiably biased: an in-depth analysis of a common criteria eal4+ certified trng, IEEE Trans. Inf. Forensics Secur. 13 (4) (2018) 1031–1041, doi:10.1109/TIFS.2017.2777342.
[8] J. Choi, Physical layer security for channel-aware random access with opportunistic jamming, IEEE Trans. Inf. Forensics Secur. 12 (11) (2017) 2699–2711, doi:10.1109/TIFS.2017.2714842.

[9] J. Tang, L. Jiao, K. Zeng, H. Wen, K.-Y. Qin, Physical layer secure mimo communications against eavesdroppers with arbitrary number of antennas, IEEE Trans. Inf. Forensics Secur. 16 (2021) 466–481, doi:10.1109/TIFS.2020.3015548.
[10] J. Laeuchli, Y. Ramırez-Cruz, R. Trujillo-Rasua, Analysis of centrality measures under differential privacy models, Appl. Math. Comput. 412 (2022) 126546 https://www.sciencedirect.com/science/article/pii/S0096300321006305, doi:10.1016/j.amc.2021.126546.
[11] S.G. Gedam, S.T. Beaudet, Monte carlo simulation using excel(r) spreadsheet for predicting reliability of a complex system, in: Annual Reliability and Maintainability Symposium. 2000 Proceedings. International Symposium on Product Quality and Integrity (Cat. No.00CH37055), 2000, pp. 188–193, doi:10.1109/RAMS.2000.816305.
[12] K. Ching-Jing, T. Hui-Chin, A revised forward and backward heuristic for two-term multiple recursive random number generators, Appl. Math. Comput. 185 (1) (2007) 240–246 https://www.sciencedirect.com/science/article/pii/S0096300306008551, doi:10.1016/j.amc.2006.06.093.
[13] R. Kawai, H. Masuda, On simulation of tempered stable random variates, J. Comput. Appl. Math. 235 (8) (2011) 2873–2887 https://www.sciencedirect.com/science/article/pii/S0377042710006643, doi:10.1016/j.cam.2010.12.014.
[14] A.M. Gergely, B. Crainicu, A succinct survey on (pseudo)-random number generators from a cryptographic perspective, in: 2017 5th International Symposium on Digital Forensic and Security (ISDFS), 2017, pp. 1–6, doi:10.1109/ISDFS.2017.7916504.
[15] P. Wang, F. You, S. He, Design of broadband compressed sampling receiver based on concurrent alternate random sequences, IEEE Access 7 (2019) 135525–135538, doi:10.1109/ACCESS.2019.2942687.
[16] A.I. Gómez, D. Gómez-Pérez, F. Pillichshammer, Secure pseudorandom bit generators and point sets with low star-discrepancy, J. Comput. Appl. Math. 396 (2021) 113601 https://www.sciencedirect.com/science/article/pii/S0377042721002211, doi:10.1016/j.cam.2021.113601.
[17] M. Haahr, Introduction to randomness and random numbers, 1999, https://www.random.org/randomness/.
[18] M.S. Turan, A. Doganaksoy, B. S., On independence and sensitivity of statistical randomness tests, in: S.W. Golomb, M.G. Parker, A. Pott, A. Winterhof (Eds.), Sequences and Their Applications - SETA 2008, volume 4, 2008, pp. 18–29.
[19] P. Hellekalek, S. Wegenkittl, Empirical evidence concerning aes, ACM Trans. Model. Comput. Simul. 13 (4) (2003) 322–333, doi:10.1145/945511.945515.
[20] P. Burciu, E. Simion, A systematic approach of nist statistical tests dependencies, J. Electr. Eng. Electr. Control Comput. Sci. 5 (1) (2019) 1–6 https://jeeeccs.net/index.php/journal/article/view/113.
[21] J.A. Karell-Albo, C.M. Legón-Pérez, E.J. Madarro-Capó, O. Rojas, G. Sosa-Gómez, Measuring independence between statistical randomness tests by mutual information, Entropy 22 (741) (2020) 1–18, doi:10.3390/e22070741.
[22] L. Fan, H. Chen, S. Gao, A general method to evaluate the correlation of randomness tests, Lect. Notes Comput. Sci. 8267 (2014) 52–62, doi:10.1007/978-3-319-05149-9_4.
[23] A. Doganaksoy, F. Sulak, M. Uguz, O. Seker, Z. Akcengiz, Mutual correlation of nist statistical randomness tests and comparison of their sensitivities on transformed sequences, Turkish J. Electr. Eng. Comput. Sci. 25 (2017) 655–665, doi:10.3906/elk-1503-214.
[24] F. Sulak, M. Uguz, O. Koçak, A. Doganaksoy, On the independence of statistical randomness tests included in the nist test suite, Turkish J. Electr. Eng. Comput. Sci. 25 (2017) 3673–3683, doi:10.3906/elk-1605-212.
[25] C.E. Shannon, W. Weaver, The mathemtical theory of communication, University of Illinois Press, Urbana., 1962.
[26] T.M. Cover, Elements of information theory, 2nd ed., John Wiley & Sons., 2006.
[27] J.A. Pardo, Some applications of the useful mutual information, Appl. Math. Comput. 72 (1) (1995) 33–50 https://www.sciencedirect.com/science/article/pii/009630039400162W, doi:10.1016/0096-3003(94)00162-W.
[28] T.O. Kvålseth, On normalized mutual information: measure derivations and properties, Entropy 19 (11) (2017), doi:10.3390/e19110631. https://www.mdpi.com/1099-4300/19/11/631
[29] R.G. Brown, D. Eddelbuettel, D. Bauer, Dieharder: a random number test suite (version 3.31.1), 2014, https://webhome.phy.duke.edu/~rgb/General/dieharder.php.
[30] G. Marsaglia, The marsaglia random number cdrom including the diehard battery of tests of randomness, 1995, https://web.archive.org/web/20160220101002/.
[31] Z. Gutterman, B. Pinkas, T. Reinman, Analysis of the linux random number generator, in: 2006 IEEE Symposium on Security and Privacy (S P'06), 2006, pp. 15pp.–385, doi:10.1109/SP.2006.5.
[32] L. Dorrendorf, Z. Gutterman, B. Pinkas, Cryptanalysis of the random number generator of the windows operating system, ACM Transactions on Information and System Security (TISSEC) 13 (1) (2009) 1–32.
[33] G. Marsaglia, W.W. Tsang, Some difficult-to-pass tests of randomness, J Stat Softw 7 (3) (2002) 1–9.
[34] P. L'ecuyer, R. Simard, Testu01: ac library for empirical testing of random number generators, ACM Transactions on Mathematical Software (TOMS) 33 (4) (2007) 1–40.

Elena Almaraz Luengo received a Mathematics degree from the University Complutense of Madrid in 2005, a Statistical Sciences and Techniques degree from the University Complutense of Madrid in 2007 and a Business and Administration degree from the National Distance Education University in 2015. She is Doctor in Mathematics from the University Complutense of Madrid since 2007 and hold a Master's Degree in Advanced Mathematics with specialization in Statistics and Operations Research, from the National Distance Education University in 2010. She is currently an Assistant Professor in the Department of Statistic and Operational Research in the Faculty of Mathematics Sciences of the University Complutense of Madrid. Her main interest are statistic techniques, probability, information security and applications.

Luis Javier Garcıa Villalba received a Telecommunications Engineering degree from the Universidad de Málaga (Spain) in 1993 and holds a Ph.D. in Computer Science (1999) from the Universidad Politécnica de Madrid (Spain).He was a Visiting Scholar at COSIC (Computer Security and Industrial Cryptography, Department of Electrical Engineering, Faculty of Engineering, Katholieke Universiteit Leuven, Belgium) in 2000 and Visiting Scientist at IBM Research Division (IBM Almaden Research Center, San Jose, CA, USA) in 2001 and 2002. He is currently Associate Professor at the Department of Software Engineering and Artificial Intelligence at the Universidad Complutense de Madrid (UCM) and Head of the Complutense Research Group GASS (Group of Analysis, Security and Systems) which is located in the Faculty of Computer Science and Engineering at the UCM Campus.His professional experience includes the management of both national and international research projects and both public (Spanish Ministry of R&D, Spanish Ministry of Defence, Horizon 2020 - European Commission, etc.) and private research projects (Hitachi, IBM, Nokia, Safelayer Secure Communications, TB Solutions Security, etc.).

Marcos Brian Leiva Cerna received degrees in Mathematics and Computer Science atUniversidad Complutense de Madrid in 2021.

Julio Hernandez-Castro is a Professor on Computer Security at the School of Computing, University of Kent, UK. He is the Deputy Director of the Kent Interdisciplinary Research Centre in Cyber Security (KirCCS). His interests are in Randomness Generation and Testing, Randomness Certification, Extractors, Distinguishers and the various applications of ML to Computer Security and Cryptography.