Masaryk University
Faculty of Informatics

The automated testing of randomness with multiple statistical batteries

Master's Thesis

Ľubomír Obrátil

Brno, Spring 2017

Replace this page with a copy of the official signed thesis assignment and the copy of the Statement of an Author.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Ľubomír Obrátil

Advisor: RNDr. Petr Švenda, Ph.D.

Acknowledgement

TODO

Abstract

TODO

Keywords

TODO

Contents

1 Introduction

- Randomness, why should we test it (defects, low entropy, etc...)
- Statistical testing of randomness

2 Used third-party statistical software

In this chapter, we will explain terminology specific to Randomness Testing Toolkit; present a quick overview of the statistical software used in the toolkit and describe the observed and unexpected behavior of the statistical software in edge cases. We also list undertaken measures to mitigate the undocumented behaviour in our further experiments.

2.1 Terminology

Throughout the thesis, we are using certain expressions in the context of Randomness Testing Toolkit and the tools and math it uses. We list these terms along with their explanations here.

(Statistical) Battery
A program developed by a third party serving as a tool for evaluation of randomness of arbitrary binary data. A statistical battery usually contains one or multiple statistical tests. The final result of the assessment is based on the results of the tests. Examples of statistical batteries are NIST Statistical Test Suite, Dieharder or TestU01.

(Statistical) Test
A single unit in statistical battery that measures some property of the tested binary stream (e.g. number of zeroes). The test can have multiple variants and subtests, and the result of the test is one or multiple statistics.

Null hypothesis - H_0
The hypothesis H_0 denotes the hypothesis that the tested data stream is random. Based on the results of the test, we can either reject H_0 and say that the data is not random or not reject it. In the latter case, we assume that the tested data is indeed random.

P-value
In our hypothesis testing, a p-value is a probability of obtaining equal or more extreme test result while H_0 holds true. In our situation, a p-value denotes the probability that we would get same or more extreme test results when testing truly random data. Therefore, the closer the p-value is to 0 the less is the probability the tested data is random and vice versa.

Alpha - α
Significance level based on which we either reject or not reject H_0. We can specify some interval (e.g. $[0, \alpha]$) and if the result of the test (p-value) falls into this interval, we will reject the hypothesis that the tested data is random. Alternatively, we can also reject p-values that are too extreme on both sides of the interval (outside of $\left[\frac{\alpha}{2}, 1 - \frac{\alpha}{2}\right]$).

Statistic
The value obtained by certain calculation from first level p-values. Multiple statistics can be obtained from one set of p-values e.g. when testing the set for uniformity we can use Kolmogorov-Smirnov Test or Chi-Square test. Based on values of statistics of a test we can decide rejection of H_0.

Test sample
A single execution of a statistical test. The result of single test execution is one first level p-value. By repeating execution of the test, we obtain multiple p-values. By using a certain statistic (e.g. Kolmogorov-Smirnov test), we can calculate a single second level p-value.

Variant of a test
Many tests can be parametrized in some ways, possibly giving different results with the same input data. We don't treat multiple executions of a single test with different settings as separate units but rather as variants of that test.

Subtest
Some tests, even when executed only once, may measure multiple properties of the data thus providing multiple results. For example, Serial Test from Dieharder battery will measure frequencies of all 1, 2, .., 16-bit patterns in the data. We treat these measurements separately - as subtests of the test. Subtests can have multiple separate statistics.

2.2 Batteries supported by RTT

2.2.1 NIST Statistical Test Suite

The battery of statistical tests was developed by National Institute of Standards and Technology (cit.). The battery implements 15 statistical tests for evaluating randomness of input data.

The reference implementation is not used in RTT because it is considerably slower than its optimized counterparts. The faster version of NIST STS used in RTT was developed by Zdeněk Říha and Marek Sýs(cit.).

2.2.2 Dieharder

Dieharder is a battery designed by Robert G. Brown at the Duke University (cit.). The battery features user-friendly console interface with the possibility of fine-grain modification of the test parameters. The fact that Dieharder is included in repositories of some Linux distributions (cit manpage) adds to its popularity and ease of use. Dieharder includes all tests from the older statistical battery Diehard (cit.), three tests from NIST STS and several other tests implemented by the author.

Since the original Dieharder implementation doesn't output all of the information we needed for interpretation and evaluation of the results, we had to modify the source code of the battery. RTT uses this modified Dieharder.

2.2.3 TestU01

This library of statistical batteries was developed at Université de Montréal by Pierre L'Ecuyer et al. (cit.). It contains a wide range of tests from NIST STS, Dieharder, and literature. It also implements various pseudo-random number generators. The statistical tests are grouped into multiple categories each intended for different use-case scenario. We will treat these categories of tests as separate batteries. TestU01 includes following ten batteries.

Small Crush, Crush, Big Crush
Small Crush battery is very fast and needs a relatively small amount of data to run - around 250 megabytes. Small Crush is also the only battery that can be natively used for analysis of data in a binary file, for the use of Crush and Big Crush, the user has to implement PRNG with the TestU01 interface. Crush and Big Crush batteries are more stringent and need gigabytes of data and a few hours to finish while Big Crush is more time and data demanding.

Rabbit, Alphabit, Block Alphabit
Tests in these batteries are suited for testing hardware bit generators and can be applied to an arbitrary amount of data. Data can be provided either as a binary file or PRNG implementing the TestU01 interface.

PseudoDIEHARD, FIPS_140_2
Tests in PseudoDIEHARD imitate DIEHARD battery; FIPS_140_2 battery implements a small suite of tests in NIST standard(cit.) Randomness Testing Toolkit doesn't support these two batteries since they are subsets of other supported batteries.

Since TestU01 is available only as an ANSI C library, we developed a console interface for it. The interface implements a dummy number generator that provides data from a supplied binary file to the battery. Our interface allows us to apply batteries to arbitrary binary data even when the batteries don't support this feature natively.

2.3 Unexpected behavior and errors of the batteries

To examine the boundary behavior of the above-listed tools, we used them to process extremely non-random data streams. The data streams that we used as the input to the batteries were two binary data files consisting of

only zeroes and ones respectively. The settings of the batteries remained set to default.

Below we list observed undocumented behavior that differs from the execution of the batteries with non-extreme input.

NIST Statistical Test Suite

Each test in the battery processed 1000 separate data streams. Each data stream was 1000000 bits long.

Tests Random Excursions and Random Excursions Variant are not applicable to all possible data streams. In a regular run with reasonably random data, this doesn't matter much, as the tests are repeated multiple times, and the final result will simply be calculated from a lesser number of p-values.

Neither of the tests can be applied to a stream full of zeroes or ones. This causes absence of results when analyzing such data. The user can find out the fact that the tests are not applicable to provided stream after he inspects logs of the program; otherwise, the interpretation of the missing results is left to him.

Dieharder

When processing extremely non-random data with Dieharder, we observed various erroneous events. The events are summarized in Table 2.1.

Test name	Stream of zeroes	Stream of ones
STS Runs	No result	No result
DAB Monobit 2	Invalid result	Invalid result
Diehard Minimum Distance (2D Circle)	Stuck execution	-
Diehard 3D Sphere (Minimum Distance)	Stuck execution	-
Marsaglia and Tsang GCD	Stuck execution	-
RGB Generalized Minimum Distance	Stuck execution	-
RGB Kolmogorov-Smirnov	Stuck execution	-
Diehard Craps	-	Stuck execution
RGB Bit Distribution	-	Stuck execution

Table 2.1: Undocumented behavior of tests in Dieharder battery

Observed errors

- **No result** Test didn't provide any p-values that would be used to calculate the final result of the test. The user is not notified of this, and the final result of the test statistic is based on default value (1.0).

- **Invalid result** Test provided resulting statistic and there was no indication of error other than that the result was again default value of the statistic (1.0). Following the definition of p-value, the interpretation of such result is that the analyzed stream was almost certainly random. This is obviously not true, as both streams are just repeating ones or zeroes respectively.
- **Stuck execution** Tests froze at a certain point in execution, did not produce any results and we were forced to kill the processes manually.

TestU01

Batteries Small Crush, Crush, Big Crush, Rabbit, Alphabit and Block Alphabit were executed. Tests that are part of multiple batteries acted in the same way across the batteries. The behavior is summarized in Table 2.2.

Test name	Stream of zeroes	Stream of ones
sstring_Run	No result	No result
sknuth_Gap	No result	-
svaria_SampleProd	All results invalid	All results invalid
svaria_AppearenceSpacings	All results invalid	All results invalid
scomp_LinearComp	All results invalid	All results invalid
scomp_LempelZiv	All results invalid	All results invalid
svaria_SampleCorr	-	All results invalid
sknuth_MaxOft	Some results invalid	Some results invalid
svaria_SampleMean	Some results invalid	Some results invalid
sspectral_Fourier3	Some results invalid	Some results invalid
sstring_HammingWeight2	Some results invalid	Some results invalid
sstring_AutoCor	Some results invalid	Some results invalid
smultin_MultinomialBitsOver	Some results invalid	Some results invalid
sstring_LongestHeadRun	-	Some results invalid
snpair_ClosePairs	Stuck execution	Stuck execution
snpair_ClosePairsBitMatch	-	Stuck execution
svaria_SumCollector	-	Stuck execution
smarsa_GCD	-	Stuck execution

Table 2.2: Undocumented behavior of tests in TestU01 library

Observed errors

- **No results** Tests reported warning and ended without any result. This is probably caused by the tests not being applicable to provided data.
- **All results invalid** All statistics of the test reported p-value very close to 1.0. This could lead the user to the interpretation that the test reports the data as an almost perfect random stream.

- **Some results invalid** Similar situation to the previous one but not all statistics of the test are close to 1.0. Results of tests statistics are either close to 0.0 or 1.0.
- **Stuck execution** Tests froze at a certain point of execution. In some cases, this is preceded by an issued warning. Tests didn't produce any results and had to be killed manually.

2.3.1 Preventive measures in RTT

Since we need to use the batteries in RTT with arbitrary binary data, we implemented following measures that mitigate above-mentioned errors in our experiments.

- Tests that don't produce any results are ignored and treated as if never executed.
- Because some tests give 1.0 as a result of their statistics when the data are clearly not random, we will reject the hypothesis of randomness either when the p-value is too close to 0 or too close to 1. More specifically, H_0 will be rejected for all p-values that falls outside of the interval $\left[\frac{\alpha}{2}, 1 - \frac{\alpha}{2}\right]$. This way we reject all results that are too extreme.
- Each test is executed with the timeout. If the test doesn't finish within defined time limit, we will automatically terminate it and then treat it as if it didn't produce any results.

3 Randomness Testing Toolkit

In this chapter we will present our developed tool, Randomness Testing Toolkit[1] and our motivation for developing the tool. The toolkit has several parts and use-cases for various types of users. The parts will be described in respective sections.

3.1 Motivation

Our main motivation for this project was to create a tool that would be easy-to-use even for users that are not experienced in field of statistical randomness testing. This includes easy testing of binary data as well as easy and straight-forward interpretation of results of the computation.

Our next goal was to create tool that would be used by CRoCS[2] members for their experiments. Main advantage of all members using unified tool and result format, is that the results are comparable and the experiments are easy to conduct.

In the toolkit we included three statistical tools that are generally used for analysis of binary data. These tools are described in Chapter 2. Each of these tools has specific interface, testing methods and result format.

Therefore, at the lowest level, RTT acts as a unifying interface to these tools. RTT accepts settings in general format and transforms them to tool-specific commands. After execution, RTT is responsible for gathering tool-specific results and transform them into easily-readable general format.

Another parts of the project allow us to easily deploy RTT to server infrastructure. Point of this deployment is to further speed up the testing. The statistical tests are sometimes time and resource demanding. Deployment on multiple servers permit us to scale the computational power when it is needed. Another advantage of this approach is that the user doesn't have to install anything on its local machine. The user needs to only provide data to be tested and choose pre-defined or provide his own configuration of the RTT.

One of the dravbacks of the remote testing is the need for data transfer. Because of this, we implemented two use cases. First use-case, intended for common user, is that he will go to a website and will upload his or her data via web browser. This is slower that direct upload to the server and viable only if the user doesn't have too much data to test.

---

1. Sometimes referenced as RTT.
2. Centre for Research of Cryptography and Security

The second use-case was created with big data testing in mind. Upon request, the user is able to gain access directly to the server via SSH and create the computational jobs via command line interface. User can access remote storage on the server and download the data for the testing. Other option is to generate the data that will be tested using utility that was also developed in CRoCS and is intended for generating various data streams from multiple cryptographic primitives.

3.2 Local RTT

3.3 Remote RTT

3.3.1 SSH

3.3.2 Webserver

3.4 Results

3.4.1 File

3.4.2 Database

3.5 Result interpretation

4 Analysis of outputs of cryptographic functions, comparison with EACirc

- How the data were tested
- List functions
- List interesting (differing) results - Dieharder, NIST STS, TestU01, EACirc, polynomials(???)

5 Analysis of DIEHARDER results on quantum random data

- Statistical intro, uniformity, first vs. second level p-value, etc...
- Two experiments - continuous p-values, blocks of 2nd level
- Results - non-uniform, where it will begin to show on 2nd level results

6 Conclusions

- Developed user-friendly tool for easy analysis of arbitrary binary data - Randomness Testing Toolkit
- Interpretation of results
- Comparison of batteries with EACirc, polynomials
- Defects in Dieharder, their relevance, etc...
- Future work, same analysis on TestU01, dependence between tests(?), continuous development of RTT, call for flawless statistical battery (:))

A An appendix

TODO