MASARYK
UNIVERSITY

FACULTY OF INFORMATICS

Improvements of the Randomness Testing Toolkit

Bachelor's Thesis

TOMÁŠ MAREK

Brno, Fall 2023

MASARYK UNIVERSITY

FACULTY OF INFORMATICS

Improvements of the Randomness Testing Toolkit

Bachelor's Thesis

TOMÁŠ MAREK

Advisor: Ing. Milan Brož, Ph.D.

Department of Computer Systems and Communications

Brno, Fall 2023


FACULTAS ARTIS INFORMATICÆ
UNIVERSITAS MASARYKIANA

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.

Tomáš Marek

Advisor: Ing. Milan Brož, Ph.D.

Acknowledgements

These are the acknowledgements for my thesis, which can span multiple paragraphs.

Abstract

This is the abstract of my thesis, which can
span multiple paragraphs.

Keywords

keyword1, keyword2, ...

Contents

Introduction

1 Randomness testing

TODO: SOMEWHERE A QUICK OVERVIEW?

The goal of this chapter is to explain the mathematical background of randomness testing to the users of the *Randomness Testing Toolkit* and make understanding of results and their deeper examination easier.

1.1 Motivation

Randomness testing is a form of empirical statistical hypothesis testing. During the test a behaviour of a data set is tested under the prediction that the tested hypothesis stands. When randomness is tested, the hypothesis states that the data are random. This means that the data contain no patterns and thus each bit has exactly the same probability of being 1 or 0.

The tested data are streams of bits usually acquired from output of various cryptographic primitives, where randomness of the output is required or expected. These are usually (pseudo-)random number generators or ciphers. The randomness testing is then used to assess the quality of the data - whether the data are random, or if the data contain patterns (i.e. are non-random).

Each randomness test detects different set of patterns, therefore more tests are usually applied on the data at the same time. Groups of tests that are applied together are called test batteries. Some groups of test might also search for similar patterns in the data, therefore their results might be correlated. TODO: SOURCE In this case it is recommended to use more than only one group of tests.

1.2 Process of testing

At the beginning of statistical testing, the *null hypothesis* (H_0), which is to be tested, is set. In the case of randomness testing the null hypothesis states, that the data are *random*. Associated with the null hypothesis is the *alternative hypothesis* (H_1), which in the case of randomness testing states that the data are not random. MOVE TO OVERVIEW Result of the test is that either we *reject* the null hypothesis (i.e. some proof

of non-randomness/pattern in the data was found, the data are not random), or the we *do not reject* the null hypothesis (i.e. no proof of non-randomness was found in the data).

Another step before we start the test is choosing the *significance level* (often denoted as α). The chosen significance level is the probability of rejecting the null hypothesis in spite of the null hypothesis being true. The usual chosen values are $\alpha = 0.05$ or $\alpha = 0.01$ TODO: SOURCE.

The next step is choosing the *test statistic* (i.e. function) of the data used for testing. This test statistic determines which kind of patterns are searched for in the current test. When choosing the test statistic, its distribution on random data must be known. Each test statistic may be a function of data of different size. TODO: SOMEWHERE MENTION THAT WE LOOK FOR TOO EXTREME SAMPLES

MAYBE MOVE TO ANALYSIS CHAPTER? We can choose from various test statistics. Most of the test statistics in widely used test batteries work with data of fixed length. TODO: REF ANALYSIS CHAPTER However, in some tests data with varying length are tested. These statistics further split into two categories. In the first category, the length of tested data is preset by user. These can be further viewed as fixed-length tests. In the second category, the length of tested data is determined during the testing process.

Once all of the previous have been chosen, the testing process begins. The process is started by obtaining data. During randomness testing these are taken from some source of randomness and are further viewed as a sequence of bits. Based on the test statistic, sequence with the required length is extracted and the rest of the data is discarded. Then the value of test statistic is computed, which is used to acquite the resulting *p-value*. For some distributions (e.g. for the standard distribution) precomputed tables are available, for the remaining distributions the p-value must be computed.

The *p-value* of the observed test statistic value is the probability of drawing a value of the test statistic that is at least as extreme as the observed result. In other words, the p-value is sum of probabilities of test statistic values that have the probability of occurring equal to or lower than the observed value.

The p-value can also be calculated as area under the probability density function of given distribution within interval of values that are at least as extreme as the observed value. The interval consists of

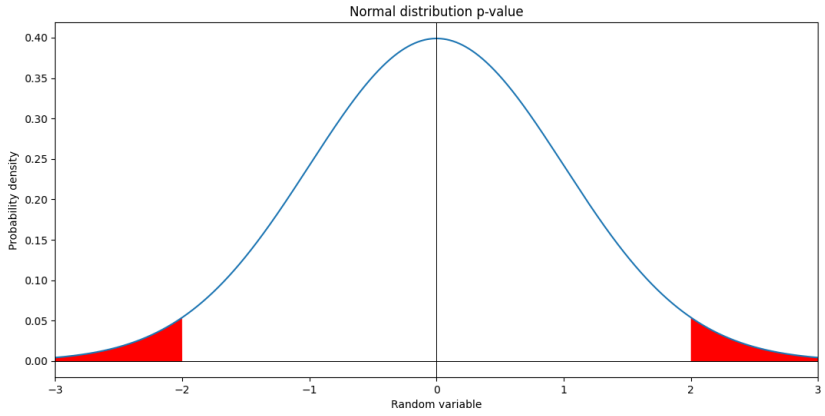


Figure 1.1: Visual interpretation of p-value

two parts. The first part is from the observed value towards infinity or negative infinity, depending on the way we gain more extreme values. TODO: SECOND PART of the interval - ask Syso

At Figure 1.1 of one such calculation can be seen. In this case, the value 2 was drawn from a standard distribution. The more extreme values are found towards the infinity (the probability distribution function is lowering), therefore the first interval is $< 2, \infty >$. TODO: SECOND INTERVAL The p-value is then equal to the area under the curve of the function within the given intervals (in red).

TODO: formula, figure

1.3 Results interpretation

The resulting p-value is then used for interpretation of the test result. If the p-value is smaller than the in advance chosen critical level, we consider the sample to come from a different distribution than is implied by the null-hypothesis. In other words, the probability of drawing this sample, given that the null-hypothesis stands, is so low, that it must have come from a different distribution. The distribution is unknown, but the probability of obtaining this sample is at reasonable level in this distribution.

TODO: SOME FIGURE WITH TWO OVVERLAPPING DISTRIBUTION

Therefore the null-hypothesis is *rejected* and we consider the tested data to be *non-random*. Otherwise, no proof of the data being non-random was found and we *do not reject* the null-hypothesis and consider the data to be *random*.

However, the interpretation of results can be more than black and white only. In general, p-values close to the critical level can be considered *suspicious*. Further analysis might then be in place.

It is important to note that this is only a *statistical* testing. Therefore it is possible that the rejected (i.e. too extreme) sample was drawn only 'by chance'. Given that the data were in fact random, the probability of this happening is equal to the chosen *critical value*.

1.4 Two level testing

TODO: mention expected fails of of first levels, figure with schema

Second level testing is used for a further examination of the data and can yield more information than using a first level testing only. For example when we examine the sequence

$$\epsilon = 00000000001100110111111111111$$

it will pass the first level test without raising any raising any suspicions of non-randomness with p-value = TODO. However this sequence clearly contains a pattern and is not random.

To perform a two level test, we split the original sequence into n equal length non-overlapping sequences. Then we perform a standard statistical test as described in Section 1.2 on each of these sequences in separate and collect all of the p-values they generated. These tests are then called first-level tests and their respective p-values are called first-level p-values.

The first-level p-values are treated as data in second round of testing, where goodness of fit test are used - the second-level test. The goodness-of-fit tests are used to determine how well the observed data fit expected distribution. The most notable examples of tests from this category are the Kolmogorov-Smirnov test and the χ^2 test. For the first-level p-values a uniform distribution with interval $< 0, 1 >$ is

expected. The second-level test yields one second-level p-value, this is interpreted the same way a one level test p-value would be (as described in Subsection 1.3).

1.4.1 Kolmogorov-Smirnov test

There are two variants of the Kolmogorov-Smirnov test (KS test). Two-sample test is used for comparing distributions of two data samples (to check if the data samples share the same distribution). The second variant, which is used in randomness testing, is the one-sample test. The one-sample test is used to compare a data sample against a theoretical distribution. In randomness testing, the observed distribution of first-level p-values is compared to the expected (uniform) distribution.

The Kolmogorov-Smirnov test is based on comparing the cumulative distribution function (CDF) of the expected distribution and the empirical distribution function (EDF) of the observed samples. The Kolmogorov-Smirnov (D) statistic is the maximal absolute difference (distance) between the expected CDF and observed EDF (as can be seen in Figure 1.2. In some variants of the Kolmogorov-Smirnov test two statistics are measured - the maximal positive difference (D^+) and the minimal negative difference (D^-) of CDF and EDF. Once the statistic is obtained, the second-level p-value is computed. TODO: FORMULA AND FIGURE

1.4.2 χ^2 test

χ^2 tests are a group of several statistical tests used for comparing two sets of categorical data. The main one is the Pearson's χ^2 test, that is used to determine existence of statistically significant difference in frequencies of categories in two sets of data. In randomness testing, one data set are the observed first-level p-values, that are divided into n equal-width categories and their respective frequencies are counted. The second data set are the expected frequencies for each category. Because the expected distribution of p-values is uniform, the expected frequencies are equal in each category.

The test is based on computing differences in frequencies for each category between both sets of data. For data with k categories the test

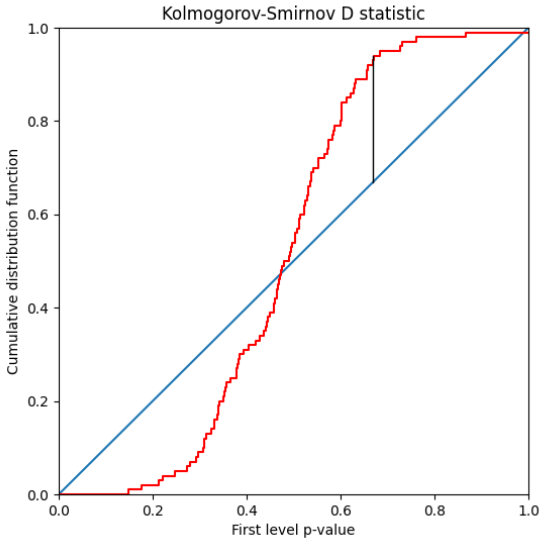


Figure 1.2: Kolmogorov-Smirnov test D statisic

statistic is defined as

$$\chi^2 = \sum_{n=1}^{k} \frac{(x_i - m_i)^2}{m_i}$$

where x_i is the observed frequency in i-th category and m_i is the expected frequency in i-th category. The p-value can be either computed or looked up in precomputed tables.

TODO: figure, mention some requirements for sample size.
mention difference between ideal continuous and real discrete

2 Available solutions

This chapter serves to describe various works and programs this thesis connects to.

2.1 Statistical testing batteries

More or less deep description of each battery. Should contain information about test parameters/settings.

If there are any problems with the battery (e.g. tests which read different amount of data from DieHarder). - Discuss colision with the paper
Mention overflow detection
how batteries interpret results (first/second level, more statistics)
strong and weak things

2.1.1 Dieharder

Explain p-samples, name tests with irregular read bytes

2.1.2 NIST STS

Explain stream-size and stream-count

2.1.3 Test U01

List all batteries. Explain repetitions, –bit -nb -w -r -s, mention repeating tests with different parameters

2.2 Testing toolkits

JUST COPIED FROM WORK TO ACADEMIC WRITING COURSE, TO BE USED AND CHANGED LATER.

In the previous chapter different randomness testing batteries were described. The typical user, however, uses more than one battery,

which means installing and running each testing battery individually. Also it is strongly recommended (sometimes even needed) to set up parameters for each test from the battery individually based on tested file and to run this test manually.

Since this approach is not convenient, Ľubomír Obrátil from Center for Research on Cryptography and Security (CRoCS) at FI MU created the Randomness Testing Toolkit (*RTT*). This toolkit allows users to run and configure three test batteries by a single command.

This work was followed by Patrik Vaverčák from Faculty of Electrical Engineering and Information Technology at Slovak University of Technology. He created newer variant of *RTT* called Randomness Testing Toolkit in Python (*rtt-py*). Compared to *RTT*, it contains two additional test batteries.

2.3 Randomness Testing Toolkit

RTT was created in 2017 and its main idea was to combine *Dieharder*, *NIST STS*[1] and *Test U01* statistical test batteries into one program. It was written in C++.

The concept of *RTT* is that it acts only as a unified interface of the batteries. Each test battery is executed by *RTT* as a separate program. The *RTT* then collects the output and processes in into a unified format. [1, p. 8]

However some problems in the processing of the output were found; these are addressed in chapter **??**.

2.3.1 Settings

The *RTT* needs to be set up by the user before running. The first part of user settings contains general settings made for the *RTT*, the second part contains configuration for individual test batteries. Each of these parts is stored in its own JSON[2] file. The original setup description is from

The general settings are stored in *rtt-settings.json* file, which has to be located in the working directory of the *RTT* [1, p. 10] . These

1. National Institute of Standards and Technology - Statistical Test Suite
2. JavaScript Object Notation

settings are usually not changed between runs. The most important setting from the general part are paths to the executable binaries of individual statistical test batteries. This is the only setting that has to be manually filled by the user.

The storage database can also be filled in by the user, but this functionality is often unused. The following general settings have implicit values and do no need to be changed unless the user wishes to. They are paths to storage directories for results and logs of individual runs and execution options (test timeout and maximal number of parallel executions of tests).

The battery configurations are dependant on the size of the tested file, therefore the file with the battery configuration is specified for each run of the *RTT*. These configurations are different for each battery (see sections **??**, **??** and **??**), but settings for all of the batteries can be stored together in a single file. [1, p. 11] The *RTT* contains several prepared battery configurations for various sizes of tested file.

2.3.2 Output

The output of *RTT* is in a plain text format. The most important part of the output is the direct report, which is saved in the results directory. At the beginning of the report are general information – the name of the tested file, the name of the used battery, ratio of passed and failed tests and battery errors and warnings in case there were any.

After the general information is a list of results of individual test runs in a unified format. The fist part of the single test report contains the name of the test and user settings (*e.g. P-sample count in Dieharder battery or Stream size and count in NIST STS battery*). The second part of the single test report are the resulting second-level P-values alongside the names of statistic used (usually Kolmogorov-Smirnov statistic or Chi-Square test). At the end of the single test report is a list of first-level P-values produced by the test. Example of the output can be seen in Figure 2.1.

```
------------------------------------------------------------
Diehard Birthdays Test test results:
    Result: Passed
    Test partial alpha: 0.01

    User settings:
        P-sample count: 65
    ************

    Kolmogorov-Smirnov statistic p-value: 0.46269520       Passed
    p-values:
        0.01554128 0.01704044 0.07338199 0.08890865 0.13047059
        0.14641850 0.14648858 0.14985241 0.15741014 0.17234854
        0.17570707 0.18313806 0.19708195 0.21929163 0.23582928
        0.23875056 0.24659048 0.24810255 0.26921690 0.29350665
        0.29444024 0.29618689 0.30017915 0.30767530 0.32816499
        0.33671597 0.33723518 0.33723518 0.35046577 0.36986762
        0.38616538 0.40739822 0.42316216 0.42606175 0.42712489
        0.46376818 0.47710967 0.51301110 0.55638736 0.58615965
        0.58816320 0.62212002 0.63106447 0.65794861 0.66078115
        0.66209483 0.67060673 0.69336319 0.70343506 0.72259414
        0.74451995 0.79749441 0.81986290 0.85442793 0.88851953
        0.88897431 0.89604503 0.92240447 0.93852901 0.93852901
        0.95468456 0.96540827 0.96785289 0.96922576 0.99790555
    ============
------------------------------------------------------------
```

Figure 2.1: The example of single test report from the *RTT*

2.3.3 Disadvantages

The problems/weak points we want to improve with this thesis. Namely at least non machine-machine readable format, running only one battery at time, maybe re-calculation of results

There are two most notable disadvantages of the *RTT*. The first one is that each battery has to be run individually by the user. This lowers the convenience of usage for the user. The second one is the output format. While it is easy to read for human users, machine reading requires complicated parsing.

2.4 Randomness Testing Toolking in Python

The Randomness Testing Toolkit in Python (*rtt-py*) was created by Patrik Vaverčák. It is supposed to be a better version of *RTT* [2, p. 24] and it was written in Python. However there are still some functional differences between *RTT* and *rtt-py*. The most notable difference is in the output format and supported batteries.

2.4.1 Settings

The settings of *rtt-py* are very similar to the original *RTT*. According to Vaverčák, the general settings from the *RTT* should be compatible with *rtt-py*, but in reality there is problem with settings for the NIST STS's experiments directory. Also, no database connection is implemented in *rtt-py*, therefore the *mysql-db* attribute is ignored. [**rtt-py-github**]

The second part of user settings are tests configurations. They use exactly the same format as those used in *RTT* (as mentioned in 2.3.1) and are interchangeable. [2, p. 25] The user has to keep in mind that the *rtt-py* uses FIPS[3] and BSI[4] batteries, which are not used in *RTT*.

3. Federal Information Processing Standards
4. Bundesamt für Sicherheit in der Informationstechnik

Results overview

	Failure rate	../input2/1000MB.dat	../input2/1000MB_2.dat
rgb_minimum_distance_0 (DIEHARDER)	0.0	0.754103	0.407390
rgb_permutations_0 (DIEHARDER)	0.0	0.931074	0.184047
diehard_operm5_0 (DIEHARDER)	0.0	0.228052	0.680182
sts_monobit_0 (DIEHARDER)	0.0	0.279259	0.096535
sts_serial_0 (DIEHARDER)	0.0	0.279259	0.096535
sts_serial_1 (DIEHARDER)	0.0	0.972893	0.052121
sts_serial_2 (DIEHARDER)	0.0	0.621721	0.618407

Figure 2.2: The example of the overview table from the *rtt-py*

2.4.2 Output

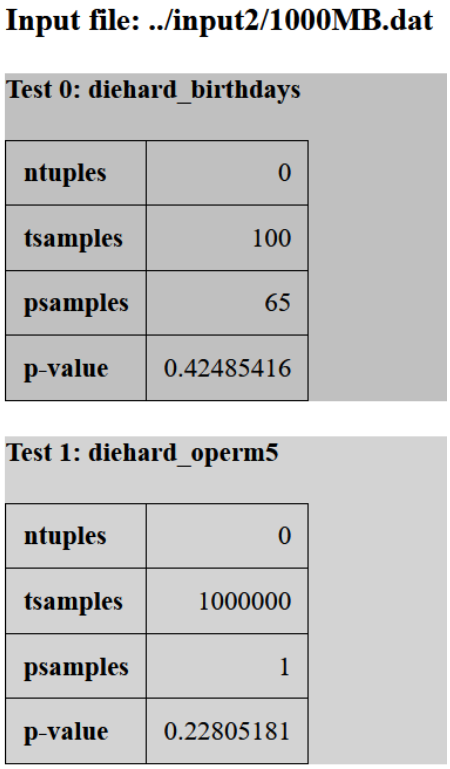
There is a significant difference in the output format between *RTT* and *rtt-py*. The *rtt-py* creates output in two formats – *CSV*[5] and *HTML*[6]. Both of these report formats contain overview table. Each row from the table represents results of one particular test. The first column contains the name of the test and the name of the battery it belongs to.

The second column contains *failure rate* - ratio representing how many instances of this particular test failed compared to number of executed instances on *all* files with data.

Each of the following columns is named after one tested file. The record contains either P-value reported by the test, or number of failed runs – this depends on the battery. Example of this table can be seen at figure 2.2.

The output in the HTML format contains more information compared to the output in the CSV format. For each battery and for each tested file an HTML file with reports is generated.

5. Comma-separated values
6. Hypertext Markup Language



Input file: ../input2/1000MB.dat

Test 0: diehard_birthdays

ntuples	0
tsamples	100
psamples	65
p-value	0.42485416

Test 1: diehard_operm5

ntuples	0
tsamples	1000000
psamples	1
p-value	0.22805181

Figure 2.3: The example of HTML Dieharder report from the *rtt-py*

In each report file there is a list of reports for each executed test from the given battery. The single test report contains the result of the test (either reported P-value, or number of failed runs) and it may contain additional information such as settings of the test or other information connected to the result. The contained information depends on the battery and on the executed test. Example of the report can be seen in figure 2.3

2.4.3 Disdvantages

One of the problems that need to be addressed is that the *rtt-py* ignores errors and warnings from tests. The most notable example why this is a problem is when the tested file does not contain enough data for current battery configuration.

In this case, the test will read some parts of the data more than once and inform the user about this situation on the error output. The test will still produce result, which will, however, be biased by repeated parts of the tested file.

This may lead to incorrect interpretation of the results and to false acceptance or false rejection of the tested data. Since the *rtt-py* ignores this, there is no way for the user to be informed about this situation.

Compared to the *RTT* the reports created by *rtt-py* contain less information. Namely the first-level P-values are ignored, even though they can be useful for deeper examination of the results and the generator.

3 Tests Analysis

3.1 Data Consumption

3.2 Time Consumption ??

3.3 Configuration Calculator

goal of the config calc, description, usage etc...

3.4 P-Values

Various problems with test p-values distributions, will probably be split into more sections

4 Implementations Comparison

4.1 Output

Mentioned differences
for both RTT and rtt-py - subset or whole?

4.2 Missing Features of *rtt-py*

4.3 Proposed improvements

included things: adding first-level p-values,

5 Conclusion

A An appendix

Here you can insert the appendices of your thesis.

Bibliography

1. OBRÁTIL, Ľubomír. *The automated testing of randomness with multiple statistical batteries*. Brno, 2017. Available also from: `https://is.muni.cz/th/uepbs/`. Master's thesis. Masaryk University, Faculty of Informatics. Supervised by Petr ŠVENDA.
2. VAVERČÁK, Patrik. *Aplikácia na štatistické testovanie pseudonáhodných postupností*. Bratislava, 2022. Available also from: `https://opac.crzp.sk/?fn=detailBiblioFormChildEBUS6&sid=D9F0BB0A8DA9926980A890D31B33&seo=CRZP-detail-kniha`. Master's thesis. Slovak University of Technology in Bratislava, Faculty of Electrical Engineering. Supervised by Matúš JÓKAY.