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

One of our goals during working in this project was to create a tool that would provide fast and user-friendly analysis of randomness even for users not experienced in the field of statistical randomness testing. Process of statistical testing generally includes finding a suitable tool, installation of said tool, execution and interpretaion of the analysis.

Another motivation was that the developed tool would unify the process of randomness analysis that is often done by reserachers in CRoCS[1]. Before the start of the project, each researcher used his own set of scripts and tools which is difficult to manage and also complicates comparation of the experiments done by different people.

Therefore we developed Randomness Testing Toolkit[2] that serves as an unified interface of the three statistical batteries presented in Chapter 2. The toolkit consists of several connected parts that are intended for various purposes. Apart from interfaces and data storage format, the parts are independent and can be exchanged if needed.

In following sections of this chapter, we will provide detailed description of the implemented parts of the toolkit.

3.1 Local unified interface

The local unified interface is at the lowest level of the developed parts of RTT. The interface is a binary executable that accepts settings and outputs the results in common format for all batteries. The executable handles transformation of general settings into the battery specific ones, running the battery executable and then transforming gathered output into general format. The process is visualized in Figure 3.1.

3.1.1 Command-line options

Will be done after RTT binary refactor.

3.1.2 Toolkit settings

The toolkit needs to have a few opions configured before execution. These options should be set only once and should be common for all subsequent runs. The toolkit settings does not modify settings of the battery that will

1. Centre for Research of Cryptography and Security
2. Also referenced as RTT or just the toolkit.

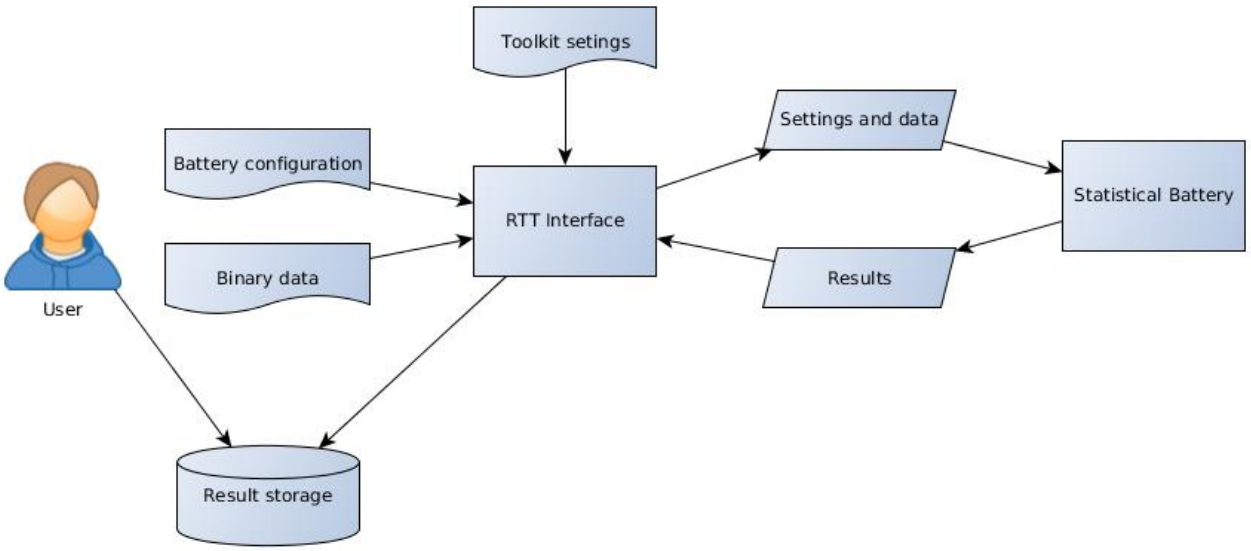

Figure 3.1: Local RTT workflow

be executed and by default are stored in file rtt-settings.json. Example of the file is shown in TODO Appendix.

The file is in JSON format and has following structure.

- `toolkit-settings` – Root tag of the file.
  - `logger` – Tag containing settings related to location of log files produced during runtime.
    - `dir-prefix` – Optional tag, if set, value of this tag will become prefix of all log locations. Can be used when all logger directories should have same parent directory.
    - `run-log-dir` – Sets location of main log file.
    - `<battery>-dir` – Value `<battery>` is replaced by values `dieharder`, `nist-sts`, `tu01-smallcrush`, `tu01-crush`, `tu01-bigcrush`, `tu01-rabbit`, `tu01-alphabit` and `tu01-blockalphabit`. All of these tags are mandatory and set locations of files that will contain raw outputs of the respective batteries.
  - `result-storage` – Section with settings related to result storages. This tag along with all child tags are optional. In case some missing options are requested by RTT, the program will issue an error. For detailed description of result storages, see Section 3.2.
    - `file` – Section with settings needed for file result storage. All values of tags except tag `main-file` are treated as directories.
      - `main-file` – Sets filename of the file with final results of the program run. If the file already exists, results will be added to it, if not, new result file will be created.

            - `dir-prefix` – If set, all directory values will have this prefix.
            - `<battery>-dir` – Value `<battery>` can be replaced by values `dieharder`, `nist-sts`, `tu01-smallcrush`, `tu01-crush`, `tu01-bigcrush`, `tu01-rabbit`, `tu01-alphabit` and `tu01-blockalphabit`. The tags set locations of files with detailed results of respective battery.
        * `mysql-db` – Section with settings needed for MySQL Database storage.
            - `address` – Address of the MySQL server with created RTT database.
            - `port` – Port on which the MySQL server is accessible.
            - `credentials-file` – Path to file that contains login information for the database. For the structure of the file see Figure 3.2.
    - `binaries` – Section with the locations of the executables of the batteries.
        * `<battery>` – Value `<battery>` is replaced by `nist-sts`, `dieharder` and `testu01`. All of the tags are mandatory. The values of the tags sets the locations of the executables of the respective batteries.
    - `miscelaneous` – Section with various settings.
        * `nist-sts` – Settings related to NIST STS battery.
            - `main-result-dir` – Sets where NIST STS stores its result files.
- `execution` Settings related to the execution of the statistical batteries.
    - `max-parallel-tests` Sets maximum number of concurrently running test processes.
    - `test-timeout-seconds` Sets time period after which the running tests will be considered stuck and will be killed.

```
{
  "credentials": {
    "username": "user",
    "password": "password"
  }
}
```

Figure 3.2: Example of file with credentials to MySQL Database

3.1.3 Battery configuration

TODO

3.2 Results

todo

3.2.1 File

todo

3.2.2 Database

todo

3.3 Remote access to RTT

Remote access to RTT allows the users to perform the binary data analysis without the need for the installation of the toolkit and without execution of the tool on their local machine. The statistical testing is, in certain configurations, demanding on computational time and resources. Having the RTT installed on remote machines allows us to scale the resources available for the toolkit and further speed-up the analysis. The user only needs to provide the data for the toolkit and choose (or provide his own) configuration for the batteries. After the testing, the user is notified and can inspect the results.

In the deployment part of the project, we developed an utility project that handles setup and installation of the local interface on single or multiple machines. The deployment project also handles setup of auxiliary scripts and tools that are required for result database and data storage that will be used by the machines used for statistical testing computation. The database and storage can be deployed on separate machines or on single server that will handle all of the tasks along with the computation of the results.

The toolkit ecosystem

- **Database server** The machine hosting database used for holding information about pending computation jobs and storing results.
- **Storage server** The machine that stores binary data that are yet to be analyzed by RTT. After the analysis the data are deleted to save storage space.
- **Backend server** Single or multiple machines that host installed toolkit and batteries. The computation of results happens on these machines. If there are multiple backend machines, the computational jobs are scheduled based on the information stored in the database. Prior to the

analysis, the data are downloaded from the storage. After the computation, the results are stored in the database and the data is deleted.

- **Frontend server** The users are supposed to interact only with the frontend server. The task of this machine is to accept requests from the users for data analysis and then transfer the data to the storage and create corresponding jobs in database. Users can access the server either using direct SSH or web interface which is more user-friendly and intuitive.

The ecosystem is visualized in Figure TODO.

3.3.1 Access via web browser

The web browser interface is intended to accomodate the needs of common users. The main idea is that the user will visit our website and, using the interface, will create his experiment. For basic testing the user is only required to upload the data and suitable battery will be chosen for him. Alternatively he can also choose the desired configuration or create his own documentation from the scratch. The results of the analysis are also shown in the interface.

This approach is not intended to be used for creating big amount of experiments.

3.3.2 SSH

The direct access to the frontend server is meant for advanced users that need to analyze big volumes of data. Upon request, the user is able to gain direct access to separated part of the system on the frontend machine. The separated system contains only tools that allows him to create experiments for RTT. The data for the testing can be uploaded from the users local machine, downloaded from another server using sftp or generated right on the frontend server. The tool that can be used for stream generation was developed at CRoCS and is capable of producing outputs of numerous cryptographic primitives. The settings of the generator provide high control of the output to the user.

3.4 Result interpretation

todo

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