Masaryk University
Faculty of Informatics

Optimisation heuristics in randomness testing

Master's Thesis

Karel Kubíček

Brno, Spring 2017

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

Karel Kubíček

Advisor: RNDr. Petr Švenda, Ph.D.

Acknowledgement

I would like to thank members of CRoCS laboratory for a friendly environment with productive seasons, joyful fun, and relaxation topics. From those, I thank EACirc team members for consulting and criticising the research presented in this thesis. Martin, Valdemar and Lubo gave me invaluable help with the thesis text. Especially I thank Petr for supervising me with his never-ending enthusiasm for science at all.

I apologise to my friends, who deserved more of my presence, but they still kept me at least a bit social. I am very grateful to my girlfriend Marta, who never left me being normal and bored by the work. Thanks to her, I never had any unused free time.

Most of all I owe to my parents, who always stand behind me. You always sensed the moments, when I need a help and surprised me with nice support. I am proud of having you, thanks.

Computational resources were supplied by the Ministry of Education, Youth and Sports of the Czech Republic under the Projects CESNET (Project No. LM2015042) and CERIT-Scientific Cloud (Project No. LM2015085) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

We also acknowledge the support of Czech Science Foundation, the project GA16-08565S.

Abstract

A detectable non-randomness of cryptoprimitive's output signals a bias of the cryptographic function. This bias may signal deeper security issues of the primitive. Therefore, statistical testing of randomness is one of the automated ways of cryptanalysis. Randomness assessment by statistical batteries is an example of such automated cryptanalysis. Research tool EACirc developed at Faculty of Informatics, Masaryk University aims to design the randomness tests, which adapts to the tested data. The tool utilises a simple heuristic based on local search. This thesis researches other metaheuristics and their influence on EACirc's success rate. In extension, proof of concept artificial neural network for randomness testing was analysed.

This thesis has three main contributions. The first is a development of a testbed of 16 well-known cryptographic functions used for randomness testing comparison. The second is an extension of EACirc by three new metaheuristics. One of them, called guided local search, outperforms all the others in terms of its success rate. The third contribution is an analysis of randomness tests produced by EACirc computation. Successful tests contain evidence of the bias in the tested data. The influence of tested metaheuristics on the complexity of these tests is analysed. It is shown that the guided local search produces the least complex tests, such that allow easier cryptanalysis.

Keywords

randomness testing, cryptanalysis, block functions, stream functions, hash functions, problem optimization, metaheuristics

Contents

1 Introduction

Cryptoprimitives, such as block ciphers, stream ciphers and hash functions are basic building blocks of secure communication. A development of such primitives is complex and time-consuming. Therefore, the primitives are expected to be supported for a long time, even extended due to backwards compatibility. Hence, vulnerabilities in them can be exploited for a long time. An example of such occasion is biased stream function RC4, supported before the version TLS 1.3 [1].

The cryptoprimitives are selected in contests such as AES [2], eStream [3], SHA-3 [4] or CAESAR [5]. Each contest consists of proposals submission, their iterative cryptanalysis and elimination until a single or multiple finalists are selected as the winners. This thesis aims at the analysis part.

There are multiple types of cryptanalysis, classified by used knowledge, required computation, and other criteria. We develop an automated cryptanalysis tool, which works on ciphertext with no prior information of the data. Our goal is not to decipher the data, but to inspect a selected property of the cryptoprimitives. For example, the properties can be following.

Bit prediction of the next bit: having ciphertext stream, the attacker's chance of predicting the next bit is around 50 %. An oracle with a higher probability of the correct guess implies the ability to build a distinguisher.

Strict avalanche criterion: every flipped bit in the plaintext changes around 50 % of the ciphertext's bits.

Semantic security: even though the plaintext usually does not look like random data, the ciphertext has to seem so. If the output would be distinguishable from random data, it may reveal information about the plaintext. This property is tested by statistical batteries and tool EACirc, which extension is one of the goals of this thesis. The previous properties can be tested as well, yet not directly.

1.1 Statistical testing

Assume encrypting many distinct, yet similar plaintexts. From the semantic security and strict avalanche properties, the ciphertext should look random. To examine it, we can use statistical batteries, consisting of many individual tests. These tests inspect statistical properties, such as the frequencies of ones and zeroes. The observed characteristic is compared with expected test statistic for an infinite random sequence. A random stream has the probability of one-half for a bit to be one, so the total frequency of ones should be similar to the frequency of zeroes.

Other tests may check different types of non-randomness, like bits dependency and patterns occurrences. A statistical battery consists of such tests. Over time, various statistical batteries were developed, such as NIST STS, Dieharder, and TestU01.

Statistical batteries are fixed set of tests. Therefore, the battery does not adapt to the tested data. The goal of the EACirc tool is to create a statistical test directly based on the tested data. To adapt to the tested data, EACirc utilises heuristic for problem optimisation. The

goal of the problem optimisation is constructing a test capable of a bias detection. The aim of this thesis is to analyse alternative optimisation techniques and implement some of them to advance the optimisation process.

Following this introduction, in Chapter 2 we explain the computation of EACirc. After that, in Chapter 3 we present methods of problem optimisation and we inspect their potential for application in randomness testing. The Chapter 4 defines analysis methodology and introduces the testing dataset. The results of the work are presented in Chapter 5, followed by comparison with the related work in Chapter 6. The Chapter 7 concludes the thesis and proposes future direction in the area. The Appendix A contains a glossary of defined and advanced terms. We emphasised these terms in the text by italic type. The Appendix C shows some additional results.

Even though the research of this work was performed mainly by myself, the plural is used in the text. EACirc and other randomness testing tools are researched by a team at the Centre for Research on Cryptography and Security (CRoCS), Masaryk University[1,2].

1. The team of randomness testing involves following people: Radka Cieslarová, Michal Hajas, Dušan Klinec, Matúš Nemec, Jiří Novotný, Ľubomír Obrátil, Marek Sýs, Petr Švenda, Martin Ukrop and others.
2. The thesis materials together with an overview of the contributions are available online on the CRoCS's pages: `http://crcs.cz/thesis/kubicek2017`.

2 EACirc

EACirc is a randomness testing tool developed at the Centre for Research on Cryptography and Security, Masaryk University [6]. The randomness is tested by an attempt to distinguish between the tested data and the reference random data. The tested data are non-random if there exists a distinguisher capable of classifying these datasets with a probability higher than a random guess.

2.1 EACirc workflow

The distinguisher is encoded as a software-emulated electronic circuit; an example is shown in Figure 2.1. The circuit consists of connectors and nodes. The nodes are arranged in layers. The first is an input layer that splits the *test vector* to bytes. These bytes are processed by further layers modifying the data by Boolean operations. The output is reduced to a single byte, upon which the circuit bases the classification.

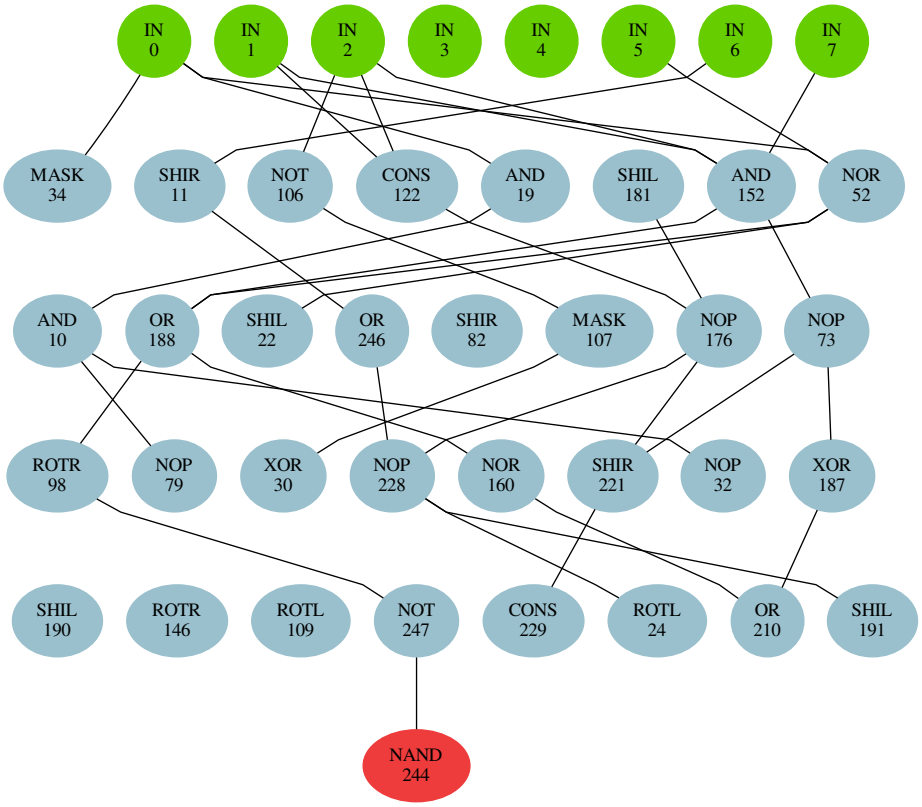


Figure 2.1: The distinguisher circuit representing a candidate solution in EACirc computation. The green layer consists of input nodes, blue are internal nodes that transform the data and the red node is output node.

The distinguisher is not able to classify a single *test vector*, as even a truly random *test vector* may look like the tested data. Therefore, EACirc process the output statistically. It proceeds 1 000 analysed data *test vectors* and 1 000 reference random data vectors, and it accumulates the outputs per dataset to an output distribution. These distributions for the tested and the reference datasets are compared using a χ^2 test [7, p. 219]. The result of the χ^2 test is a p-value, which is transformed to the *fitness* value as $1 - p$-value.

There is an enormous number of potential circuits. Therefore the computation starts with a randomly generated circuit, generally denoted as a *candidate solution*. To obtain intended solution, we perform modifications, which are addition or removal of a connector or change of the operation in the node. By these changes that we denote as *mutations*, we can create another *candidate solution*, also sometimes denoted as a *neighbour*.

These two *candidate solutions* are compared using the *fitness* function. We want to minimise the p-value, which means maximising the *fitness*. Therefore we select the *candidate solutions* with the higher *fitness*. We repeat the process 100 times; these iterations are called *epoch*.

If the *epoch* is too long, the circuit may adapt to much to the current dataset, instead of generalising the non-randomness property. Such situation is called *overfitting*. Therefore, we change the tested dataset and start a new *epoch*. The *epoch length* determines the learning process's potential.

The whole process is called *iterated local search*. It is a metaheuristic from a list of single-solution metaheuristics.

3 Metaheuristics

A computational problem is a task to find a solution for the given question. An optimisation problem is a task to find the feasible solution, which satisfies the criteria best. A heuristic is a problem specific technique for finding some approximative solution, where exact techniques fail. Finally, the metaheuristic is technique abstracted from the problem, so that it can be used for various optimisation problems.

Metaheuristics are applicable for optimisation problems if a programmer can provide three crucial parts. The first is an instance of candidate solution for the problem; it is the circuit in EACirc. The second is a function to vary the solution, in EACirc it is the mutation. The third and usually the most important is the *fitness function*, which measures the quality of the solution. Metaheuristics use these parts to follow the landscape of the problem, and they try to find better solutions during the time.

The main advantage of metaheuristics is that they are well-known and studied. This overview, as well as implementation ideas, are following book Metaheuristics from Talbi [8]. Please refer to this book for more examples of metaheuristic applications. The citations that are mentioned in this thesis extend the literature of the book, mainly in areas close to an application in cryptography.

The number of candidate solutions inspected simultaneously splits metaheuristics into two categories: single-solution metaheuristics are present in Section 3.1 and population-based metaheuristics in Section 3.2. Section 3.3 describes other methods. All the approaches are firstly described and then analysed for the application in randomness testing. If the application is suitable, we describe the possible implementation in more detail.

3.1 Single-solution metaheuristics

Figure 3.1 shows the classification of the improved variants of local search metaheuristic.

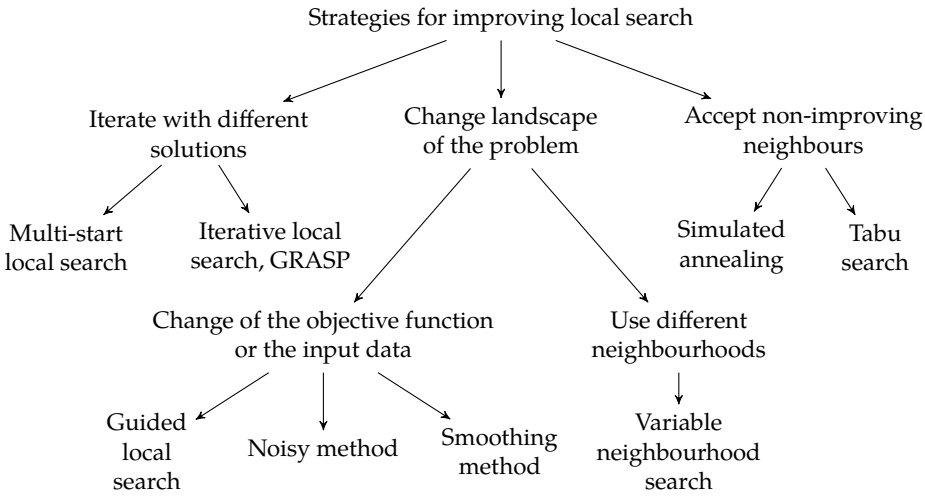


Figure 3.1: Single-solution metaheuristics as specialized version of local search. The source: [8, Figure 2.24, p. 125].

3.1.1 Local search

Local search is the very basic approach in the problem optimisation. The search begins with a single initial random candidate solution, which is mutated into a *neighbour* candidate. Then the local search compares *fitnesses* of these two solutions and selects the better candidate to the next generation. The algorithm iterates this approach until it finds a satisfactory solution, or it ran for the limited number of iterations.

The algorithm can also be referred as hill-climbing, as we with acceptance of an improving solution, we step up in the *fitness* space. Therefore local search often finds only the suboptimal solution – a *local optimum*.

Escaping *local optima* is an important phenomenon, as *fitness function* landscape depends on tested data, or on the specific evaluation approach.

As EACirc is using an iterated version of local search, the application of local search would be straightforward. However, this would decrease the capabilities of EACirc do to omit of the learning process.

3.1.2 Iterative local search

Iterative local search is and extension of local search. After defined amounts of iteration of basic local search, the tested data are changed. The computation on one set of data is called an *epoch*, so we denote the number of iterations *epoch length*.

During the *epoch*, the *fitness function's* progress is non-decreasing. However, after the end of each *epoch*, the *fitness* is expected to drop. The final *fitness* has to be tested on a new dataset. This rule applies for every metaheuristic. Therefore, we distinguish between the learning dataset and the testing dataset.

The *epoch length* is a crucial parameter of this metaheuristic. A short *epoch length* can be computationally more demanding due to higher data consumption, and the solution may not be able to adapt to the data. Too long *epoch* leads to *overfitting*, a situation when the individual is learning a specific pattern of the current dataset over general patterns of all datasets. Martin Ukrop examined the *overfitting* process of EACirc in [9, Chapter 7].

The EACirc has used the iterated local search since [10]. Therefore, it is used as a baseline for single-solution metaheuristics tested in this work.

3.1.3 GRASP

The greedy randomised adaptive search procedure is splitting each iteration into two steps: a feasible solution construction by greedy algorithm and performing the local search using the generated solution.

We refused this metaheuristic as we did not found any greedy algorithm for circuit construction, such that would reflect tested data. Also, it does not fit the iterative process of learning in *epochs*.

3.1.4 Multi-start local search

Multi-start local search is another approach how to surpass *local optima* to find the *global optimum*. The metaheuristic starts the computation with k randomly generated individuals that are concurrently but independently optimised by the local search.

This parallelism could increase the usage of the testing data in EACirc. The disadvantage is a more difficult evaluation, as Ukrop stated in [9, Chapter 5]. It is a question if we should evaluate the best, the average, a random solution, or more solution simultaneously. Another possible advantage of multi-start local search can be parallelization of the computation. However, EACirc computation is already parallelized, neither on the level of single computation. Due to statistical interpretation explained in Chapter 2, we have to repeat the computation process 1 000 times.

We decided not to cover multi-start local search, as saving a small fraction of the computation time for a more complicated interpretation is a bad trade-off.

3.1.5 Guided local search

Guided local search is a single-solution metaheuristic based on dynamic changing of the *fitness function*. If the metaheuristic got trapped in a *local optimum*, guided local search would change the *fitness function*, so the individual can continue climbing.

The EACirc computation can get trapped in a *local optimum* mainly in two cases:

1. the individual got lucky to guess current test set, but it will not be successful in the next *epoch*, or
2. the individual is already successful due to finding feature of the tested data, but it could not get better.

We do not need to fix the first case, as such individual would be replaced in the next *epoch*. However, we can use guided local search to improve individual from the second case. Our alternative *fitness function* can be designed to reflect the density of the circuit. Such *fitness* would simplify the solution, which would help us with the interpretation of the last individual. Alternatively, we can use completely different *fitness function*.

We decided to implement guided local search in EACirc, as it also represents single-solution metaheuristics with a change of the objective function from Figure 3.1.

3.1.6 Noisy method

The noisy method is another metaheuristic, which changes the *fitness* space to escape *local optima*. It is done by adding random noise to the input data, as Talbi described in [8, Section 2.9.2]. At the beginning of the computation, the noise is the strongest, and it disappears during the computation so that the last *epochs* are evaluated on the unmodified data.

In EACirc, our research goal is already distinguishing pseudo-random data from reference random data. An additional random noise in the input would probably lead to an even more complicated learning process. For example, the noise could flip a bit on a po-

sition, which is necessary for distinguishing between the data sources. Therefore, this method was refused.

3.1.7 Smoothing method

Smoothing method is the last metaheuristic using changes in the *fitness function* to escape *local optima*. The method is based on smoothing the landscape of *fitness function* by computing a local average of all *neighbours* to given distance. If the *fitness function* is overall increasing in one direction, but more zoomed observation shows small fluctuation, smoothing method smooths the fluctuation and allows locating the *global optimum*.

Neighbours in EACirc are defined as solutions obtained by *mutation* of the circuit. Available *mutation* methods are an addition or a removal of a connector or changing the Boolean function in the node. The *neighbourhood* of an individual in EACirc is enormous (over 2^{25} *neighbours* in the distance one for the basic EACirc setup), so this method would not be possible due to computational requirements.

3.1.8 Variable neighbourhood search

Variable neighbourhood search also changes the landscape of the problem. However, it does not modify the *fitness function* or its interpretation, but it defines other methods for *neighbour* selection. If the computation gets trapped in a *local optimum*, the *neighbourhood* method is changed. Alternatively, the *neighbourhood* can be changed periodically. This method does not guarantee to escape from the *local optimum*, as all *neighbourhoods* can contain only worse solutions.

Specifying different *neighbourhood* in EACirc is complicated. However, the method lead us to a different idea. EACirc is heading two mutually exclusive problems. We would like to have circuit connected sparsely for easier interpretation of the final solution. However, we also want to have the circuit's input and output nodes connected. We can solve both problems by reduction of the searched *neighbourhood*.

If we already have a successful solution, we can inspect only solutions with fewer connectors. In contrast, if the circuit is not connected (so the circuit's *fitness* is equal to zero), we can inspect only the individual with more connectors.

3.1.9 Simulated annealing

Remaining two single-solution metaheuristics can accept non-improving *neighbours*. Short term acceptance of worse solution can help to escape from *local optima* and finding the *global optimum*.

Simulated annealing metaheuristic is inspired by the physical process of annealing in metallurgy. The annealing is an iterative process of heating the metal to increase the size of the crystals, which reduces the probability of a defect. During the process, the metal is heated to lower and lower temperatures.

Simulated annealing uses variable temperature, which describes the probability of worse *neighbour* acceptance. This probability is decreasing during the computations. In the be-

ginning, the likelihood of acceptance of worse solution is high, but at the end of the computation, only better solutions are accepted.

This metaheuristic is widely used, which was the reason for implementing it as a testing scenario for metaheuristics. Besides, it serves as a represent of the group of single-solution metaheuristics accepting non-improving solutions. Its implementation in EACirc does not require complex changes, only the specification of this method.

3.1.10 Tabu search

Similarly to simulated annealing, tabu search can also accept non-improving solutions to escape from *local optima*. However, the decision whether to accept a non-improving solution is not stochastic, but it is based on knowledge of already visited solutions. Therefore, tabu search holds a memory of visited solutions, called *tabu list*, and it would not use twice the same solution. This way, tabu search avoids cycles during the optimisation process.

Application of tabu search in EACirc has several issues. Firstly, how to store used individuals (memory usage may be a problem), which can be solved by storing only a hash of the solution. Secondly, the *neighbourhood* space of an individual is huge, as stated in Section 3.1.7. It is higher than the number of individuals tested during whole computation. Choosing twice the same solution is unlikely. Therefore, we decided to try only simulated annealing and leave the decision of tabu search implementation on the result of simulated annealing.

3.2 Population-based metaheuristics

Population-based metaheuristics are usually inspired by biological processes. The group of evolutionary algorithms is motivated by the process of biological evolution, which treats a population of species and forms better individuals. The ant colony optimisation and swarm intelligence algorithms are based on a motion behaviour of a colony of selected species.

Population-based metaheuristics are usually more complex than single-solution metaheuristics. The complexity occurs on multiple levels.

- The implementation usually requires many configurable parameters, so it is more prone to programming mistakes.
- The fine-tuning of these parameters can be sensitive to the selected problem.
- The algorithm needs more computational resources due to overhead with maintenance more complex method.

However, population-based metaheuristics bring significant advantage. The population together may prevent the algorithm from ending in a *local optimum*, as long as the metaheuristic maintains well the diversity of the population.

3.2.1 Evolutionary algorithms

Evolutionary algorithms are a group of population-based metaheuristics inspired by the natural selection in the evolution of the species. They are the most spread and studied population-based metaheuristics in computer science beginning in the 1980s. Evolutionary algorithms can be used to solve both continuous and discrete problems, and they are also used for both single- and multi-objective problems.

The computation is an iterative loop of three main steps, together denoted as a generation. The evolution starts with a random population of initial solutions.

Selection of parents from the population. Usually based on the *fitness function* with some randomization technique.

Reproduction to new offsprings using *mutation* and *crossover* – algorithms for recombination of the parents.

Population replacement by offsprings. Here can be again used the evaluation by *fitness*.

This iterative approach can by an application of well-selected *fitness function*, *mutation* and *crossover* increase the quality of the individual. The algorithm can be stopped by the number of generations or by achieving target *fitness*.

During the 40 years, many variants of evolutionary algorithms emerged.

Genetic algorithms traditionally use bit representation of the individuals. They apply both *mutation* and *crossover* to form new individuals. The selection of new population from the offsprings is stochastic.

Evolutions strategies are used in continuous optimisation, so the individuals are represented by real values. The main offspring formation method is the *mutation*; the *crossover* is rarely used. Evolutions strategies apply elitist selection to form a new population.

Evolutionary programming is used in series prediction problems and continuous optimisation. It emphasises *mutation* and does not use recombination. The population is selected stochastically.

Genetic programming evolves programs as solutions. They are usually expressed as a tree (like abstract syntax tree). Both *mutation* and *crossover* are applied. Even though genetic programming is the least studied approach of evolutionary algorithms, and it is computationally expensive, it is often used in classification problems.

Randomness testing is a discrete optimisation problem; therefore, we may apply genetic algorithms and genetic programming. Genetic algorithms were used in multiple works for analysis of bias in ciphertext from round reduced TEA and DES functions [11–15]. On the other side, we found no successful work on the same topic using genetic programming.

3.2.2 Ant colony optimisation

The ant colony optimisation is an instance of swarm intelligence metaheuristics. This metaheuristic imitates the cooperative behaviour of ants. Every ant has a simple goal of collecting food, and the common goal is to find the shortest path between the food and the nest. The motion of the ants is coordinated by their smell and a pheromone.

- Every ant spreads pheromone on its trail.
- Ant prefers to take a path with more pheromone.
- The pheromone evaporates over time.

The common application of ant colony optimisation is as shortest path algorithms. The literature states applications in travelling salesman problem, for example [16]. However, the metaheuristic can also be applied to combinatorial problems, as the problem representation can be encoded to the graph structure and the shortest path algorithm may solve them. An example of such application can be formula satisfiability problem (SAT) [17].

The problems solvable by ant colony optimisation are usually problems, where the solution is constructed from building blocks. The pheromone can be associated with the block, and it can describe the probability of using the block in the final solution. However, EACirc's circuit cannot be decomposed into independent components, so we do not test this metaheuristic.

3.2.3 Scatter search

The scatter search method is highly utilising the combination of the individuals. The algorithm holds a population of the best so far inspected individuals called a *reference set*. Like every population-based metaheuristic, the diversity of the *reference set* is crucial. Scatter search is an iterative algorithm, where one iteration consists of four steps.

1. The *reference set* is split into subsets of the size two to four individuals.
2. We combine the individuals of the subsets forming new candidate individuals.
3. We run a single-solution metaheuristic improving the candidate individuals.
4. The *reference set* incorporates the candidate individuals based on the *fitness*.

The main issue of application is the second step, where the solutions are combined. We would have to develop a function, which combines multiple solutions and affects their shared properties. However, there is no guarantee that various advanced individuals share the same property. Also, the approach is similar to evolutionary algorithms. Therefore, we did not test it.

3.2.4 Other swarm intelligence methods

The swarm intelligence metaheuristics adopt a collective behaviour of decentralised and self-organized systems. Similarly to ant colony optimisation, the behaviour of the swarm can be used to solve optimisation problems. Swarm intelligence fits the best problems close to the swarm behaviour. Therefore, the main application is in decentralised control of robots, droids and other artificial individuals.

The application issues for our problem are similar as for ant colony optimisation.

3.3 Other methods

There are various optimisation methods, which are not classified as single-solution or population-based metaheuristics. However, those models or algorithms are still capable of learning of classification.

The classification is important in machine learning field. The field also includes algorithms for the interpretation of the results, detection of outliers, learning on a low amount of data or non-complete data. Because of these advantages, we decided to experiment with machine learning models.

These models and algorithms can be used for various goals. Two main are feature selection and classification. Our principal aim is the data classification, while we expect, that the model itself will select the features. The model has to adapt to the data. Therefore, we cannot make manual feature selection.

3.3.1 Artificial neural networks

The artificial neural networks (ANN) are classification optimisation method inspired by the process of learning of brain and neural system. The initial idea was inspired by the research of Donald Hebb [18], who described the neurones and the process of learning of living beings' neural systems. The Hebbian learning arose several trivial nonsupervised learning methods. Further development came with the invention of backpropagation algorithm by Paul Werbos [19]. The last rapid increase of usability of artificial neural networks came in 2006 with significant speed-up both of the algorithms and their implementation on GPUs. Both together allowed deeper networks and more analysed data due to parallelization, leading to rapid progress in scalability and usability of artificial neural networks. Please refer to Goodfellow's Deep Learning textbook [20] for more details about artificial neural networks.

The artificial neural network consists of purpose specialised layers. The first layer is called an input layer, and its input are the data for classification. Then there may be multiple hidden layers, which sequentially process the output of previous layers. The data proceed to the last – output layer, which classifies the data. Each layer consists of neurones. Each neurone takes as input the output from all neurones of previous layers, multiplied by weight for each connection. The output is processed by an *activation function*, which ensures its normalisation. There are many types of *activation function*, and each layer can use different *activation function*. The most used *activation function* is a sigmoid function, with a real number input and the output is the interval $(-1, 1)$ or $(0, 1)$.

The artificial neural networks were used to solve various problems in classification and pattern recognition. The most common are applications in image processing, signal processing, text data processing, robotics, and system control.

In cryptography, we have seen fewer applications of ANN. The main area of application of ANN in cryptography is authentication [21], as it is a classification problem.

An application in encryption is even rarer. For example, researchers from Google Brain showed an experiment where they let two neural networks develop encryption function for communication. Another neural network was intercepting the communication and at-

tempting to decrypt it [22]. Preceding works [23, 24] did not get such attention as Google's work, but they had a similar objective with simpler assumptions. However, all these works are far from application in cryptography. The only inspiration from them to our project is normalisation of binary data to floats, that was done by encoding zero as -0.5 and one as $+0.5$. We have found no further relevant application of ANN in the classification of binary data.

We decided to utilise the best practices of learning in classification by ANN. However, the application of ANN in our scope is not straightforward. We do not know in advance, what are the features in the data. Therefore, the network has to have the whole ciphertext as an input. We encode the binary input to floats (0 as -0.5 and 1 as $+0.5$) based on intuition supported by other works [22, 23]. We tried both fully connected network (where every node is connected to every node of previous layer or every input bit), and convolution network, where the first layer has only a limited local interval of inputs.

3.3.2 Other statistical classification methods

There are several other models or algorithms of statistical classification. Those are for example support vector machine (SVM), decision tree with a C4.5 algorithm, *k*-nearest neighbours algorithm (*k*-NN), Bayes classifier and many others. Those methods usually rely on good feature selection, which we do not perform in our scenario. Therefore, the application of those models was not examined within this work.

4 Experiment methodology

The primary testing goal of EACirc is to assess randomness of ciphertext of cryptographic functions. As most of the cryptographic functions are in a scope of automated analysis semantic secure, EACirc can identify non-randomness only for reduced functions. The functions can be reduced for example in a number of rounds, by processing selected plaintexts and keys, and by selecting only specific bits of the output. As the analysis of reduced versions of cryptographic functions is the main goal of the EACirc project, it is also selected as the testing scenario for comparison of analysed metaheuristics.

The approach for evaluation is briefly described in Chapter 2. The tested data were produced from 16 different cryptographic functions. These functions are reduced by the number of rounds. We selected intervals of rounds based on EACirc reference capabilities, such that the EACirc is capable of distinguishing function reduced to i rounds, but it is not able to distinguish $i + 1$ rounds. Usually, we test also $i - 1$ and $i + 2$ rounds, for broader comparison of the results with state of the art of statistical testing cryptanalysis.

EACirc works as a distinguisher between the tested dataset (produced by the cryptographic function) and the reference random data. Formerly we used *quantum random number generator* (*QRNG*) services as the source of the reference data. The need of having *QRNG* data for the computation has the following disadvantages.

- Transmission of the reference data on the computation grid brings a slowdown and a more complex handling of the computations.
- The potential of an error with the data manipulation. *QRNG* are not error-less, for example, Quantis device we tested in our laboratory produced biased data (see results on [25]). Detection of such biases is common use-case of randomness testing.
- More requirements on EACirc's users and developers. We have to ship the tool with the reference data.

Therefore, we switched from *QRNG* to *pseudo-random number generator* (*PRNG*) PCG32 [26] as reference data generator. PCG32 is fast and computationally strong *PRNG*. We did an experiment with comparison PCG32 to *QRNG*, and we found no bias. We also retested old results using *QRNG* newly with *PRNG*, and the results are the same. Hence, we believe built-in *PRNG* is a more secure and convenient option.

4.1 Used data

Former works on EACirc analysed candidates of cryptoprimitives competitions [27–29]. These works extended EACirc by many functions sharing the same interface. This approach was effective for broad analysis of the competition candidates, but it does not test well-known cryptographic functions like AES and DES.

Therefore, for the comparison with other randomness testing tools, we decided to select a new test set of cryptographic functions. We chose the finalist of competitions SHA-3 [4] (BLAKE, Grøstl, JH, Keccak and Skein) and eStream [3] (HC-128, Rabbit, Salsa20, SOSE-

MANUK and Grain), and above that, we implemented other block (AES, DES, 3-DES and TEA) and stream (RC4) functions.

4.1.1 Reimplementation of stream handling in EACirc

The implementation of SHA-3 and eStream candidates in EACirc was called projects. The projects were handling everything related to data generation, specifically IV, key and plaintext. Every project had its own implementation of the *counter* plaintext generator, the *strict avalanche criterion* generator, and other data sources.

For the implementation of the new project for block ciphers, we decided to reimplement data generation in more variable and unified way. For this, we decided for data manipulation via streams, which can be nested into each other. All streams inherit from an abstract class `stream` with a minimalistic interface. The streams have to specify its `size`, and they have to implement method `next()`, which returns the next *test vector*. We have the following types of streams in EACirc.

- Data sources streams are used as an input to cryptographic functions or as reference data. An example of such streams are:
  - input streams: true and false bits generators, counter and multiple variants of the *strict avalanche criterion* generator, and
  - *PRNGs* PCG32 [26] and Mersenne Twister [30].
- The projects of cryptographic functions. Namely project eStream and SHA-3 and newly implemented block ciphers.
- Postprocessing streams are applied to data from cryptographic functions for more advanced experiments. They allow us an analysis of selected bits of the function.

The streams are completely configurable via `JSON` configuration file. As the streams can be nested, the structure is described as a `JSON` tree. The configuration file update changed the file's structure. Therefore, the configuration files are not backwards compatible. Nevertheless, the configuration is explained in examples in the project's documentation [31].

All these changes significantly simplified the codebase, unified many implementations of the same functionality and broadened the experiment capabilities of EACirc. Therefore, it was the main feature for EACirc 4.1 release.

4.1.2 Implementation of functions

AES, DES and 3-DES implementations follow educative repository [32]. The author of this repository verifies the encryption to the *test vectors* to ensure correctly working implementation. The only modification of the code is a limitation of the functions in the number of rounds.

TEA implementation is based on Wikipedia code [33], and it was already implemented in EACirc before. The implementation was simplified and reduced due to changes in the interface. It is also reduced to the selected number of rounds.

RC4 is also based on the educative repository [32], but the code is unchanged as RC4 has no round structure.

All these functions together with the new interface of new block project and the *test vectors* processing algorithm were released in minor version EACirc 4.1.

Corrections and code review of SHA-3 and eStream functions were another part of the streams update. Multiple cryptographic functions from the previous implementation were incorrectly round-reduced.

- SHA-3 Skein, eStream Rabbit, SOSEMANUK and F-FCSR were not round-reduced, but their implementation uses an iterative approach, so we reduced them.
- eStream HC-128 stated in the header file and the project documentation, it can be round-reduced, but it had no effect on the function at all.
- eStream Salsa20 used a wrong implicit number of rounds for default settings.

These bugfixes together with a fix of an improper configuration of eStream project were one of two major reasons for releasing EACirc 4.2. All these fixes together with valid configurations are described in project documentation [31].

4.1.3 Data generator tool

For direct comparison of the tested methods, we need to analyse the same data. EACirc has a built-in generation of the *test vectors* feeding the computation, as is described in Section 4.1. Therefore, we implemented a side project from EACirc for the production of data. Within the thesis in denoted as a *generator* and it shares the source code of cryptographic functions with EACirc. The codebase sharing is done using GitHub submodules, so the *generator* has its repository [34].

The *generator* is fully configurable via `JSON` configuration file as EACirc. The structure of the file is reduced, as the configuration describes only generated data. So the *generator* inherits all the capabilities of streams, providing a complex control on the data production. For simplification and automation, we also prepared a script generating configuration files for the most common settings, which allows a simple execution by command line arguments. For even simpler access to testing data, we generated 100 MB file for each basic setting per each widely-known cryptographic function (described in Section 4.1.2) reduced to a wide interval of rounds. The plaintext for this data was the counter stream. This dataset and broader description how it was generated are publicly accessible from *generator's* repository [34].

4.2 Methodology for specific optimisation methods

As the tested scenarios vary not only by optimisation method but by used tools as well, the methodology changes for given metaheuristics. The base of the experiment was used for single-solution metaheuristics, as they were implemented directly to the EACirc and they inherited the experiment environment of our tool.

For direct inter-method comparison, we need to use the same data for testing. Therefore, all used data are produced by the *generator*, described in Section 4.1.2. The cross-method comparison is still sensitive to the testing approach, so we describe it in detail in following sections.

4.2.1 Single-solution methodology

All single-solution metaheuristics were implemented into the EACirc tool. Therefore, this part extends the explanation of the single run evaluation from Chapter 2.

The main advantage of single-solution metaheuristics is their simplicity and straightforward interpretation. However, the overall interpretation of EACirc results became complex, as we work with randomness that needs to be evaluated statistically. The complex approach also leads to significantly better results. The statistical evaluation, as well as distributed computation automation, are described below.

Statistical evaluation

As was stated in Section 2.1, the *fitness* of an individual is $1 - p$-value of two sample χ^2 test, which tests, if the output bytes per source have the same distribution. For a well-working individual on a non-random data, the *fitness* is close to 1. As the quality of the individual decreases, the *fitness* decreases as well. However, completely random guess like coin tossing has the p-value uniformly distributed on the interval $(0, 1)$.

As the p-values are uniformly distributed for a random guess, getting high p-value can imply two scenarios. Either the χ^2 test rejected the null hypothesis that the distributions are independent, and it scored a significant output. Alternatively, we were lucky on seemingly random data, and the test returned high p-value from a uniform distribution on the interval $(0, 1)$. Therefore, we avoid interpretation of a single p-value. We collect a single p-value per every *epoch* computed on a new dataset. We process these p-values by a Kolmogorov-Smirnov test at the end of the run. The Kolmogorov-Smirnov test analyses the uniformity of usually 300 p-values. This test is using *significance level* $\alpha = 1\,\%$, so it rejects the null hypothesis that tested data are random, in a case they are random, in 1 % of runs. To avoid such case, we run the experiment with the same setting 1 000 times, and we analyse the *rejection rate* given as the number of failed KS tests per 1 000 runs. If the *rejection rate* is around 1 %, the data seem to be random. If it is significantly higher, the experiment rejected randomness hypothesis of the tested data.

This approach is vastly increasing the amount of data needed for the computation, as well as the overall run-time. This method is a trade-off for the success rate of EACirc, as different evaluators do not need statistical evaluation. Hence they need much fewer data; however, such evaluators are slightly worse based on the results from [35].

Considering the basic setting for single-solution metaheuristics, EACirc consumes 4.8 GB of ciphertext of tested function (Figure 4.1). The runtime of the experiment on MetaCentrum varies from 4 to 20 hours on a single core.

$$\Sigma = 1\,000\ \frac{\text{runs}}{\text{experiment}} \cdot \left(300\ \frac{\text{epochs}}{\text{run}} \cdot 1\,000\ \frac{\text{vectors}}{\text{epoch}} \cdot 16\ \frac{\text{bytes}}{\text{vector}}\right) \approx$$
$$\approx 4.8\,\text{GiB per experiment}$$

Figure 4.1: The amount of tested data analysed by EACirc for a single configuration of randomness testing experiment. A single run also evaluates data randomness, however, the capability of EACirc decreases in such mode. A single result is prone to incorrect evaluation due to statistical errors.

Computation automation

This part contains implementation and running details about the experiment's workflow. It is not crucial for understanding the main goal of this thesis – the evaluation of multiple metaheuristics in randomness testing. It documents the experiment's workflow for the purpose of replication.

As EACirc execution needs to be repeated on MetaCentrum grid infrastructure, the automation for binary deployment, computation distribution, aggregation of the results and their postprocessing is required.

The scripts are executed in the following steps.

1. Firstly, a preprocessor creates necessary configuration files for the experiment.
2. Secondly, MetaCentrum scripts distribute the computation among the grid infrastructure.
3. Thirdly, the postprocessor script aggregates the results after the computation.
4. Finally, the table generator script generates a human readable version of the results as well as JSON dump of the whole experiment. The whole process is shown in Figure 4.2.

The EACirc project formerly used tool Oneclick created by Ľubomír Obrátil [36]. The Oneclick was a single tool for both preprocessing and postprocessing phase, and the run was parametrized. However, this approach was programmed for BOINC infrastructure being outdated for the MetaCentrum. The advantages of new scripts are:

- they are single purpose, so it is easier to modify one without affecting other,
- they are written in bash (running scripts for MetaCentrum scheduler) and Python (processing), so they do not need to be compiled,
- they are maintained up-to-date for the newest EACirc,
- they are faster, as MetaCentrum computation do not require formerly used redundancy procedures.

These scripts are expected to be modified by individual researchers for the needs of their experiments, so the scripts are stored in GitHub repository [37], and users are supposed to fork them. The repository also contains `readme.md` file specifying the first steps of computations on MetaCentrum with these scripts. Every script also contains documentation in the code (concerning expected modifications for individual purposes), and when it is run without arguments, it prints out a help message.

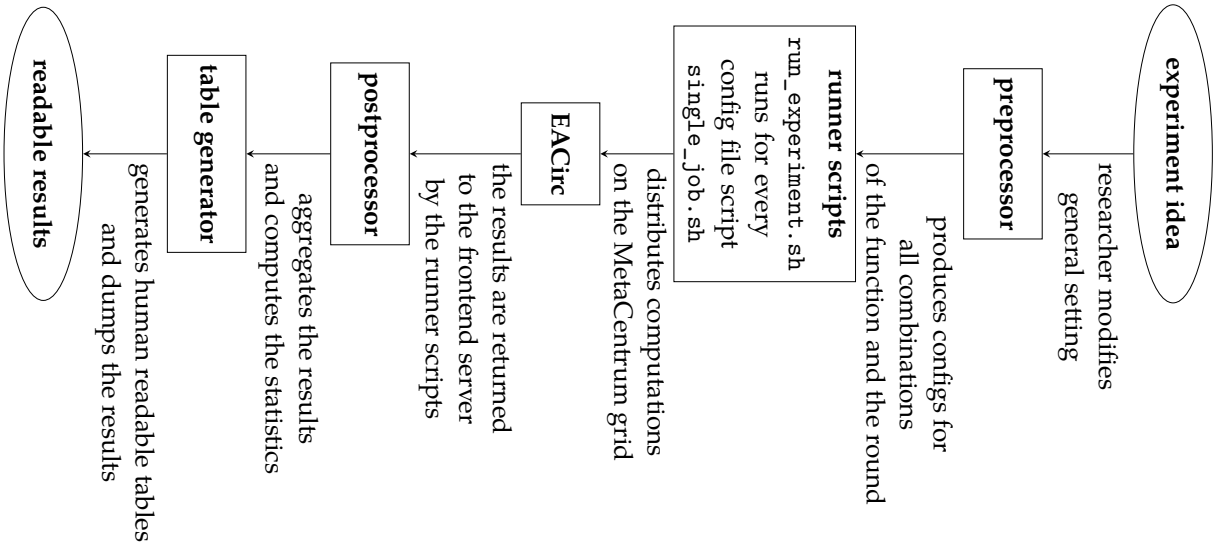


Figure 4.2: Execution of EACirc on MetaCentrum grid – order and purpose of the scripts.

Preprocessor EACirc computation is based on a configuration file with specification and environment settings for a particular experiment. As the test suite contains tens variations of parameters, it is necessary to automatize config files creation, so the user would not make a mistake during this process. The script is written in Python.

Running scripts The scripts for distribution of the computations among the MetaCentrum network are a result of Martin Ukrop's thesis [29]. They are continuously maintained in `EACirc-utils` repository [37] for the current EACirc version as well as the MetaCentrum updates.

As each experiment needs to be run multiple times for statistical evaluation, so the scripts repeat the computation in a loop with different seeds usually for 1 000 times. The same scripts then collect the results of the computation.

The current version of these scripts is also published on the GitHub [37], and the forked repository is ready to run experiments in the usual way.

Postprocessor As the computations are repeated for statistical significance, the postprocessor script has to aggregate results from many produced files. It searches through output files and collects the results. Afterwards, it calculates an aggregated statistics of the experiment.

The output of the postprocessor is a `JSON` dump of the results, which is used both for human readable results generation and also storing the results for later usage.

Table generator The generator takes as an input the `JSON` dump of the results from the postprocessor. Currently, it only generates a LaTeXtable with highlighted results, but it can be extended for more complex manipulation with the results.

$$\Sigma = 100 \frac{\text{epochs}}{\text{experiment}} \cdot 25 \frac{\text{vectors}}{\text{epoch}} \cdot 16 \frac{\text{bytes}}{\text{vector}} \approx \begin{matrix} 40\,\text{kB for learning} + \\ 40\,\text{kB for evaluation} \end{matrix}$$

Figure 4.3: The amount of data analysed by neural network for a single configuration of randomness testing experiment.

Goal specification

The ultimate goal of the experiments with metaheuristics is increasing detection capabilities of non-randomness of tested datastream. The analysed functions, described in Section 4.1, are round-reduced. The randomness of the data increases rapidly with the rounds addition, so the comparison of metaheuristics may not be fine-grained enough. We would need functions with a low increase of the randomness per round as a benchmark, but we did not observe such behaviour by any of the tested functions, nor our own design of benchmarking function does not behave that way [38].

Less ambitious, but still useful goal can be an increase in the *rejection rate* of the null hypothesis per experiment. Rejecting only 10 % of runs lead to difficulties with the analysis of the output test. The higher the *rejection rate*, the easier is the interpretation of found distinguisher.

Another goal can be decreasing the computation time or the amount of used data. As we usually fix the *epoch length* to 100 iterations, the data usage and CPU time are directly related. Moreover, as the amount of used data is already high in this settings of EACirc, the time comparison is more valuable than consumed data comparison.

We can also analyse specific goals per metaheuristics. Both variable neighbourhood search and guided local search set a complementary target – a simplification of the circuits for later analysis.

4.2.2 Artificial neural networks methodology

The neural network also classifies the *test vectors* into two sets – the tested data set and the reference data set. Therefore, the implementation takes as input two binary files – the tested file and the reference file. We generate these files with the *generator*, so we ensure consistency and comparability of the results of the approaches.

The execution of the neural network consists of a training phase, where we are iteratively changing the batches of the learning data, and an evaluation phase, where we use fresh data for the final evaluation of the network's *success rate*. The whole computation needs significantly fewer input data than EACirc, as is shown in Figure 4.3.

We prepared automation scripts for an easier analysis of multiple files together. We have a script for sequential execution of all the runs for all the binary files in the given directory, which produces the results in `JSON` format. This file is afterwards proceeded by a script, which plots the learning progress and final *success rate*.

5 Results

To verify every implemented method, we perform a *sanity experiment* comparing random data to random data. Such a test gives us a reference *critical value*, under which we do not reject randomness hypothesis. As this reference *critical value* depends on the analysed method, we state it explicitly with every test scenario.

In Section 5.1 we show the results of single-solution metaheuristics, followed by Section 5.2 with an analysis of metaheuristic's influence on the complexity of produced distinguisher circuit. In Section 5.3 we show the results of artificial neural networks and finally in Section 5.4 we compare all the methods introduced in this thesis.

5.1 Single-solution metaheuristics

Single-solutions metaheuristics use evaluation described in Section 4.2.1. It means, we execute EACirc 1 000 times independently with the same setting and we evaluate how many of the runs rejected the randomness hypothesis. The number of rejections divided by the number of runs is called the *rejection rate*. As we use a *significance level* $\alpha = 1\,\%$, the *sanity experiment* should reject the randomness hypothesis on average in 1 % of the runs. As we would like to have higher confidence in this *sanity experiment*, we perform it with 100 000 runs. The probability of incorrect *rejection rate* of such case can be computed from the binomial distribution. For example, a *rejection rate* higher or equal to 1.2 % of the *sanity experiment* with 1 000 runs can occur with the probability of 4.787 %, while the probability decreases to less than 0.001 % for 100 000 runs. We include the measured *rejection rate* to the first row of further presented result tables. Other rows contain results of round-reduced cryptographical functions.

Table 5.1 and additional Tables C.1 to C.3 in the Appendix summarise results of the tested single-solution metaheuristics applied in EACirc. Every row of the table corresponds to one cryptographical function. The lower index after the function name specifies the number of rounds we limit the functions to in EACirc. The columns with -1, 0, +1 and +2 denote an offset to the number of rounds specified in the index. For example, AES_3 was tested reduced to two rounds, three rounds as default, and four rounds. Five rounds were not tested, which is denoted by a dash in the corresponding cell. The number in the cell represents the *rejection rate* for the particular case. For indistinguishable data, values around the *rejection rate* of the *sanity experiment* are expected. Results higher than twice the *rejection rate* of the *sanity experiment* can happen in less than 0.15 % cases for indistinguishable data, we denote it as *empirical critical value*. Therefore, results over the *empirical critical value* are not considered random.

We use red highlight to ease the comprehension denoting that EACirc detected non-randomness. Otherwise, EACirc has not found any evidence against the randomness of the tested data. Therefore, we cannot say anything about the randomness of the data (we can neither reject, nor accept their randomness hypothesis). We highlighted untested scenarios (dash in the cell) if the results were estimated from prior works [9, 10, 35, 39] or faster test by statistical batteries. All the functions behave as expected; their randomness never decreases with an increasing number of rounds.

Function\rounds	-1	0	1	2
rnd_rnd$_{\text{no rounds}}$	–	0.01128	–	–
AES_3	1.0	0.160	0.007	–
$BLAKE_1$	1.0	0.110	0.021	0.006
$Grain_2$	–	1.0	0.009	0.011
$Grøstl_2$	–	1.0	0.010	0.010
HC-128$_{\text{no rounds}}$	–	0.008	–	–
JH_6	–	1.0	0.012	0.008
$Keccak_3$	1.0	0.017	0.008	–
$MD6_8$	–	0.774	0.008	0.011
$Rabbit_0$	–	0.017	0.007	0.015
RC4$_{\text{no rounds}}$	–	0.011	–	–
$Salsa20_2$	–	1.0	0.010	0.008
Single DES_4	1.0	0.204	0.001	0.010
$Skein_3$	1.0	1.0	0.016	0.010
$SOSEMANUK_4$	–	1.0	0.017	0.011
TEA_4	1.0	0.444	0.012	0.013
Triple DES_2	–	1.0	0.006	0.007

Table 5.1: Results of iterated local search as EACirc metaheuristic on the well-known cryptographic functions. The values in the cells are the rejection rate of the randomness hypothesis. The structure and the values of this table are explained in Section 5.1.

5.1.1 Iterated local search

As was stated in Chapter 2, EACirc used iterative local search as the metaheuristic since version 4.0. Therefore, measuring this metaheuristic serves as a baseline allowing direct comparison of other metaheuristics. In addition, it can be used to estimate comparison to prior EACirc results [9, 10, 35, 39], which were verified by this work.

The results of iterated local search are shown in Table 5.1. From these results, we can conclude the following regarding the testbed.

- The *empirical critical value* (the result in the first row) is significantly (as is computed in Section 5.1) higher than 1 %. It is caused by a known, but yet an unresolved bug, described in the documentation [40]. This bug is a consequence of unconnected input to output in the circuit, so we denote it as the *connection bug*. We tried fixing the *connection bug* using many techniques. However, none of them was statistically sound, and none has fixed the issue globally.
- The randomness of three-round AES, one-round BLAKE, eight-round MD6, four-round DES, and four-round TEA is in the interpretation described in Section 4.2.1 rejected. However, the *rejection rate* is less than 100 %. Therefore, the probability of obtaining a working distinguisher is lower than 100 %. We denote these results as *benchmarking functions* results, as the quality of different metaheuristics directly affect them. Therefore, these *benchmarking functions* provide a highly sensitive comparison of the metaheuristics.

We also performed other tests, which showed some additional information.

- It is possible to obtain the *rejection rate* higher (up to full 100 %) than shows Table 5.1 for the *benchmarking functions*. However, advancing the results require time-consuming manual fine-tuning to the tested data or more computational time. In the study of TEA [39], we showed almost 100 % distinguisher for four-round TEA. Obtaining such result required different shape of the circuit due to TEA block length and 100 times more *test vectors*. On the other hand, an increase in the *rejection rate* by better metaheuristic applies in general, so all results should benefit such improvements.
- The results shown in Table 5.1 were tested with the *epoch length* of 100 generations. It means that the metaheuristic performed 100 generations on the same dataset. This setup seems to lead to *overfitting*, as Ukrop examined in [9, Section 7.1]. We tried decreasing the *epoch length* to 20, which resulted in the increase of influence of the *connection bug*, as the *empirical critical value* raised to 0.03127. Such setup did not increase the number of distinguished rounds for any function, but it increased the *rejection rate* of all the *benchmarking functions*. However, only the *rejection rate* of three-round AES increased significantly to 0.982. The same behaviour appeared for shorter *epochs* for all metaheuristics.

5.1.2 Simulated annealing

We tested the simulated annealing as a representative of single-solution metaheuristics that can accept a non-improving solution. The results are available in the Appendix as Table C.1, as they are very similar to Table 5.1 with iterative local search results. A direct comparison of notable metaheuristics results on the *benchmarking functions* will be summarised in Section 5.4.

The *sanity experiment* ended with the *rejection rate* of 0.01681, which corresponds to a higher influence of the *connection bug* for metaheuristics that accepts non-improving *neighbour*. As the *rejection rate* of the random-random experiment increases, all other *rejection rates* increase as well. Therefore, the *connection bug* probably influenced *benchmarking functions* results, so the increased *rejection rate* of them may not imply a significant improvement.

The simulated annealing uses two configurable variables that influence the learning process. Those are the initial temperature and the cooling ratio. They both corresponds to the probability of accepting a non-improving solution. We analysed three different cooling scenarios, every on three different initial temperatures and cooling ratios. The cooling scenarios are illustrated in Figure 5.1 and the difference is based on the behaviour on changed learning data. The tested variables were $(\textit{initial temperature}, \textit{cooling ratio}) \in \{(500, 0.9), (100, 0.98), (50, 0.995)\}$. Every combination of cooling scenarios and temperature variables were analysed. The Table C.1 is the final and the best result of those nine tested configurations. The configuration is described by the blue temperature scheduling curve with the $\textit{initial temperature} = 500$ and the $\textit{cooling ratio} = 0.9$, as suggests Talbi in [8].

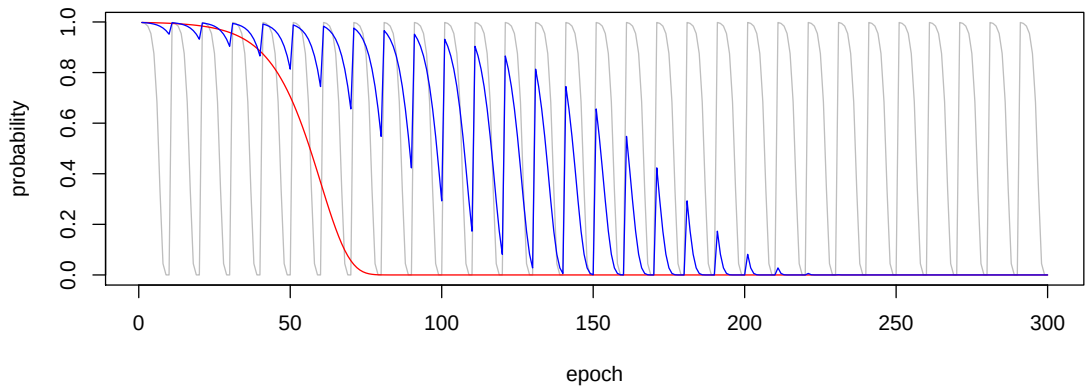


Figure 5.1: The probability of accepting a non-improving solution for simulated annealing metaheuristic. Three colours show three tested scenarios of cooling schedule. Red line shows slower cooling schedule, which is based only on the current iteration. Remaining scenarios involved the change of tested data at the end of epoch. The blue curve illustrates the best-observed scenario.

5.1.3 Guided local search

We implemented second *fitness function* for guided local search. It is equal to the number of connectors in the circuit. Within the guided local search scope of the text, we refer to this second *fitness function* as *penalisation*.

Guided local search can escape *local optima* by the change of the evaluation method. Our goal was set to escape not *local optima*, but unwanted solutions with *fitness* equal to 0, which is caused by *connection bug*. We *penalise* unconnected solutions to prefer denser circuits. This *penalisation* proved to lead to more connected solutions faster.

In contrast, solutions with *fitness* higher than 0.9 are *penalised*, if they have too many connectors. We expected that the circuit of strong distinguisher would be sparser, which was also verified by the experiments. The both *penalization* methods were tested independently, and they both have a similar role in the improvement of the results of *benchmarking functions*. However, they are working especially well together, as they hold the circuits reasonably dense.

The *penalisation* for increasing connectivity can be referred to as *space exploration*, while the *penalisation* for reducing connectivity corresponds to *solution exploitation*, concepts well known from machine learning. For further explanation, please refer to Talbi [8, Section 1.3.3, page 24].

The final results of the guided local search are similar to results of iterated local search Table 5.1. Therefore, they are available in the Appendix in Table C.2.

Unrelated fitness functions

We have also analysed the original idea of guided local search, such that we should change between unrelated *fitness functions* to help the metaheuristic find a better solution. We im-

plemented another *fitness function* called *weight evaluator* based on different statistics than the χ^2 test. While the χ^2 test analyses the bits and their order in the circuit's output byte, the *weight evaluator* analyses only the values of the bits, ignoring their position. Therefore, the *weight evaluator* can be compared to the *monobit test* of circuit outputs. Hence the *fitness function* using *weight evaluator* alone performs slightly worse than χ^2 test *fitness function*.

Concurrent usage of these both *fitness functions* resulted in significantly worse distinguishing capabilities than the first method described in this sub-section.

5.1.4 Variable neighbourhood search

The goals of the variable neighbourhood search are very similar to the guided local search goals. We also want to increase the connector density in the case of unconnected circuits and decrease it if the circuit works as a good distinguisher. The guided local search ensures this through the *fitness function* changes, while variable neighbourhood search changes inspected individuals. If the circuit is not connected, it tries only denser circuits. If the circuit is working well, it considers only sparser circuits for the *neighbours*.

If we have an unconnected circuit, we try adding a new connector, so the probability of reaching connected graph increases. In the opposite case, when we have a good working individual, we examine only solutions with fewer connectors.

The variable neighbourhood search results are in the Appendix in Table C.3. It is performing better than the iterative local search. However, overall it performs slightly worse than guided local search, as can be seen at the end of this chapter in inter-approach comparison in Table 5.5. It is probable that the limitation of the *neighbourhood* does not allow exploration of a significantly better solution in case of well-working solutions.

5.2 Analysis of the resulting circuit

EACirc's run produces a final distinguisher circuit. If the distinguisher can detect non-randomness of the tested data, its structure can provide an insight into the source of this non-randomness.

For the analysis of the circuit, we had to prepare a visualisation of the output circuit. EACirc dumps the final circuit to a file in `dot` format, which is transformed to image via Graphviz visualisation tool [41].

We also perform *pruning* of the circuit removing the unnecessary connectors and nodes. It is implemented as a BFS traversal of the circuit from the output node. This way, we can eliminate all connectors and nodes that are not affecting the output. *Pruning* works better for sparse circuits, where it can reduce the number of connectors by more than 50 %. Dense circuits can sometimes be pruned by less than 10 %.

5.2.1 Guided local search

A secondary goal of the guided local search was the simplification of output circuits. When the metaheuristic finds a distinguisher for the tested data, it *penalises* dense circuits over sparser circuits. During this circuit simplification, guided local search does not lose

any relevant connectors, as the leading *fitness function* is still the χ^2 test. Therefore, we remove only unnecessary connectors.

The output from a run using guided local search created a significantly sparser circuit, usually already with less than 50 % of connectors compared to iterative local search runs, as is shown in Figure 5.2. The *pruning* of such circuits resulted in very sparse circuits. These simpler circuits are easier for analysis, so it is less demanding to find the source of non-randomness of the tested data. *Pruning* is an important tool for manual cryptanalysis. The cryptanalysts can reveal potential weaknesses of the cryptographic function based on a simple circuit. The visualisation and *pruning* also help our team understand the EACirc computations.

Pruning works only for individuals that are good distinguishers. If the tested data seem random, guided local search will not create sparse circuit – circuits remain even more connected. The density of the circuit can also show how successful a distinguisher is.

5.2.2 Variable neighbourhood search

As in Section 5.2.1, we inspected the density of the output circuit. Again, the circuits were on average sparser than circuits from iterated local search. However, they were significantly denser than circuits from guided local search runs. The observation may also support the hypothesis that limiting the *neighbourhood* is too strict, and disallows valuable changes in the learning process.

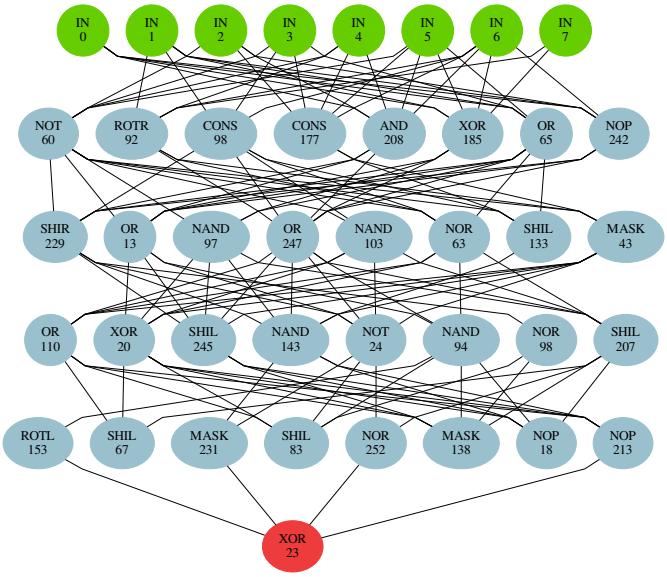
5.3 Artificial neural networks

We analysed artificial neural networks on the same dataset. The algorithm takes two binary files as an input – the tested data and the referential random data. The first step is encoding the data into a `NumPy` array of `floats`. Then we form two mutually exclusive datasets – the learning and the testing dataset. Both contain randomly chosen *test vectors* from either of the input files, together with the label of the data source.

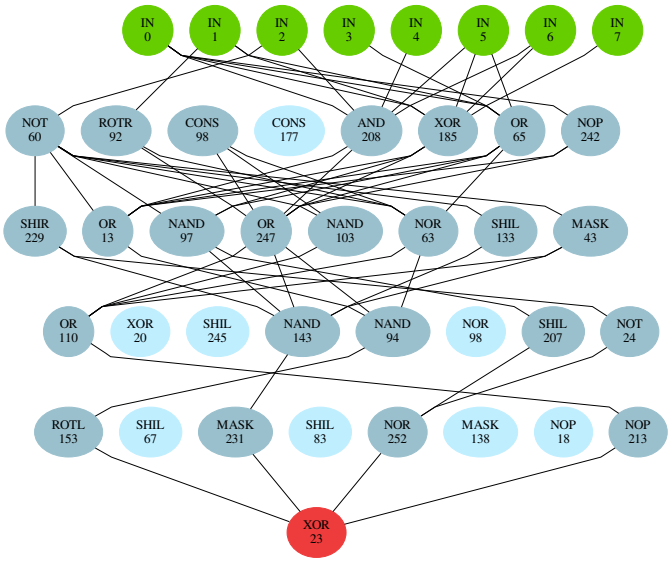
We implemented the neural network in the Python library Keras [42], which is a high-level API for TensorFlow [43]. Keras allows a simple prototyping of the network's layout, while TensorFlow is a highly effective implementation of linear algebra for neural networks capable of computation on CPU and GPU. The implementation is entirely independent of the EACirc, and the code is published on GitHub [44].

We initially designed the network's layout by intuition, trial and error and available tips for networks layout from the best practices of image processing. We designed a network with five layers layout. The input layer's width depends on *test vector's* length. Usually, it is 128 bits, encoded as `floats`. Then follow three hidden layers with widths eight, four, and two neurones. The output layer consists of a single neurone.

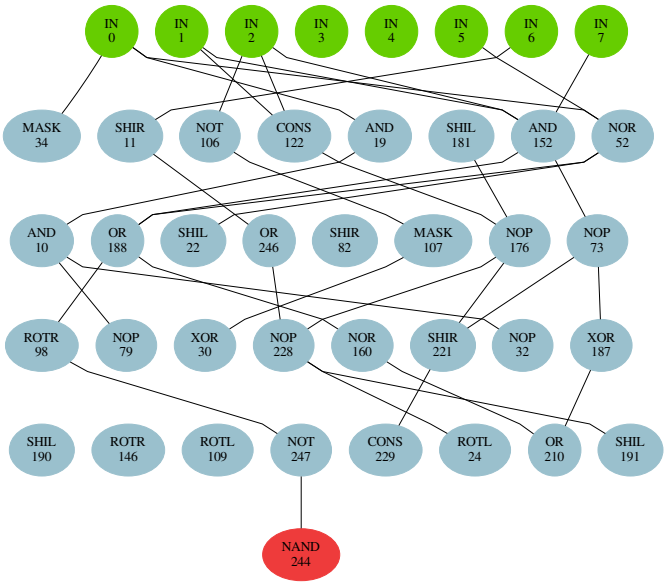
We analysed the performance of such prototype network and selected a benchmarking function for the comparison of the influence of network's parameters. We tested multiple layouts. The layout of the prototype network can be noted by a number of neurones in the hidden and output layers – 8-4-2-1. We experimented with following layouts: 1, 16-1, 8-4-2-1, 8-8-8-1, and 16-8-4-2-1. The second tested parameter of the network was the number



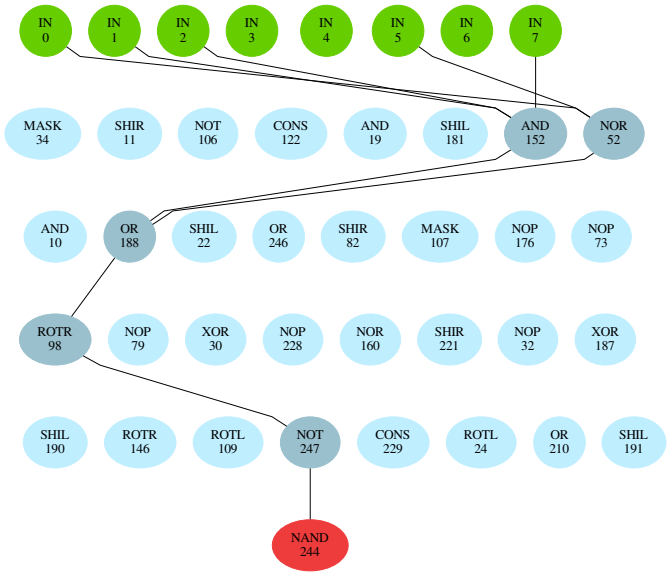
(a) An iterated local search circuit before pruning.



(b) An iterated local search circuit after pruning.



(c) A guided local search circuit before pruning.



(d) A guided local search circuit after pruning.

Figure 5.2: The top two circuits were produced by iterated local search, while bottom two by guided local search. The tested data for all the cases were four-round DES. While the both unpruned and pruned iterated local search circuits are quite dense, even the unpruned guided local search circuit is sparse. However, if the tested data seems random, the guided local search circuit would be even denser than Figure 5.2a.

of *test vectors* evaluated in a single iteration. Our preliminary results on the prototype network showed that smaller batches lead to better results. We analyse two properties of the execution: what was the final *acceptance rate* of the network and how fast did the network learned the first statistically significant distinguisher. The *acceptance rate* is equal to the ratio of $\frac{\#_{\text{correct guesses}}}{\#_{\text{total guesses}}}$.

Interpretations of the observations from Tables 5.2 and 5.3 are the following.

- An increasing batch size decreases the learning speed, as can be seen in Table 5.3. Our reasoning for this observation is that more testing vectors per one iteration overload the network with too much information. An ideal batch size is within 10 to 50 *test vectors*.
- Complex networks learn slower than simple networks. Sometimes the complex networks are not able to learn from the data at all. Nevertheless, the network of a single neurone performs always the best over all the examined batch sizes. The single neurone can still observe if the *test vector* contains more ones than zeroes as well as simple patterns.
- The learning capabilities of the neural networks in our implementation are limited. Based on these observations, we cannot expect neural network is performing better than EACirc.

We evaluated one of the best working parametrisation (single neurone and batch size 25) of the neural network on the well-known functions dataset. The results are in Table 5.4 where rows and cells are the same as in the previous results in Table 5.1 and in the cell is a final acceptance rate of the run. The *acceptance rate* is equal to the ratio of $\frac{\#_{\text{correct guesses}}}{\#_{\text{total guesses}}}$. It measures the quality of the ANN, similarly as a *fitness function* was the metaheuristics' metric. The red highlight has the same meaning as before, the *critical value* used for rejection of randomness was set to 0.55, as such results have the probability lower than 1 %.

We also experimented with convoluted neural networks (CNN). The convoluted neural network works excellent for image and signal processing, as the convoluted layer detects local patterns. However, cryptographic functions should meet the *strict avalanche criterion* suppressing any data locality. The CNN serves as a specific test for locality-specific patterns. If the reduced cryptographic function meets the *strict avalanche criterion*, the CNN should perform worse, as the first layer of CNN has lower capabilities than a fully connected layer of ANN. Overall, the CNN performed worse than ANN. The main problem was slower learning than ANN.

Layout\batch size	**10**	**25**	**50**	**100**	**250**	**500**	**1000**
1	0.985	0.967	0.957	0.934	0.937	0.641	0.716
16-1	0.992	0.970	0.960	0.935	0.938	0.665	0.719
8-4-2-1	0.525	0.663	0.510	0.581	0.605	0.505	0.500
8-8-8-1	0.515	0.632	0.736	0.617	0.652	0.549	0.562
16-8-4-2-1	0.516	0.500	0.503	0.507	0.504	0.506	0.518

Table 5.2: The final acceptance rate of the artificial neural network on two-round Salsa20 for different parametrizations. Rows describe the used layout and columns represents the size of the batch.

Layout\batch size	10	25	50	100	250	500	1000
1	1	1	0	1	2	10	9
16-1	2	2	1	6	4	31	20
8-4-2-1	–	53	–	–	96	–	–
8-8-8-1	–	76	47	76	63	–	–
16-8-4-2-1	–	–	–	–	–	–	–

Table 5.3: The parametrizations analysis of the neural network on two-round Salsa20. The cell contains the iteration number when the acceptance rate overcome 0.6. Rows describe the used layout and columns represents the size of the batch.

5.4 Inter-approach comparison

The interesting details about single-solution metaheuristics are summarised in Table 5.5. The artificial neural network models are omitted in the table. They are not able to detect non-randomness of functions reduced to the given number of rounds.

- Guided local search is always more successful than the baseline iterated local search, and it is almost always the best among the tested methods. The single better result of simulated annealing does not mean it is better than guided local search, due to the higher *empirical critical value* caused by the *connection bug*. The same holds for four-round TEA and variable neighbourhood search, as it has a longer computation time.
- The ANN can detect bit-level non-randomness, while EACirc cannot. The bit-oriented evaluation is expected to to be crucial for the cryptanalysis. However, this did not lead to the creation of ANN distinguisher for any function, that would EACirc not be able to distinguish.
- The runtime of the simulated annealing and the guided local search is the same as the runtime of the iterated local search. The variable neighbourhood search can take up to five times longer. The data consumption of all metaheuristics in EACirc is the same, as described in Figure 4.1.

  In comparison, the neural networks need significantly less data (as was shown in Figure 4.3). One run of ANN takes about ten times less time than one EACirc run. However, due to varying capabilities, these numbers are not comparable. EACirc can also find distinguishers for the two-round AES within less than ten *epochs*, even though we set 300 *epochs* as the default. Interesting comparison of the efficiency of EACirc implementation is that C++ EACirc consumes twelve times more data per second of computation than ANN implemented in Python.

  Still, the ANN is the only method, which can be used to classify a single *test vector* (16 bytes). The framework provides both its classification guess and the estimation of its certainty in the guess. No other method provides such application. EACirc is not able to classify a single vector in the current setting due to the statistical evaluation described in Chapter 2. Nevertheless, EACirc can be set to evaluate a single *test vector*. Such setting was used by Ukrop in [9].

Function\rounds	**-1**	**0**	**1**	**2**
$rnd\_rnd_{\text{no rounds}}$	–	0.503	–	–
AES_3	1.0	0.501	0.498	0.506
$BLAKE_1$	1.0	0.500	0.501	0.503
$Grain_2$	1.0	1.0	0.500	0.508
$Grøstl_2$	0.496	0.500	0.498	0.495
$HC\text{-}128_{\text{no rounds}}$	–	0.497	–	–
JH_6	1.0	1.0	0.495	0.501
$Keccak_3$	0.973	0.498	0.505	0.500
$MD6_8$	0.677	0.492	0.509	0.505
$Rabbit_0$	–	0.499	0.493	0.497
$RC4_{\text{no rounds}}$	–	0.504	–	–
$Salsa20_2$	0.776	0.770	0.499	0.499
Single DES_4	0.637	0.508	0.502	0.509
$Skein_3$	0.999	0.577	0.505	0.508
$SOSEMANUK_4$	1.0	1.0	0.496	0.501
TEA_4	0.750	0.507	0.514	0.504
Triple DES_2	0.999	0.656	0.507	0.500

Table 5.4: The best-obtained results of ANN on well-known cryptographic functions dataset. The values are corresponding to acceptance rate, where value 0.5 is expected for runs that cannot reject the randomness hypothesis of the tested data. The dash represents a scenario that was not tested.

$Function_{rounds}$\Metaheuristic	**Iterated local search**	**Simulated annealing**	**Guided local search**	**Variable neighbourhood search**
AES_3	0.160	**0.305**	0.164	0.182
$BLAKE_1$	0.110	0.051	**0.260**	0.157
$MD6_8$	0.774	0.419	**0.995**	0.992
Single DES_4	0.204	0.093	**0.496**	0.441
TEA_4	0.444	0.244	0.729	**0.877**

Table 5.5: The rejection rate for selected cryptographic functions of EACirc utilising different single-solution metaheuristics.

6 Related work

We presented our tools for randomness assessment; yet this area is broad, and many researchers developed their own tools. However, these tools are not usually adaptive to the data as EACirc is. Instead, they test the data for a set of statistical properties. These statistical tests are grouped to statistical test batteries. There are multiple batteries; we present the following ones.

Donald Knuth explains statistical testing in the second edition of The Art of Computer Programming [45]. He suggested several *PRNGs*, and for analysis, he designed some statistical tests. The high quality of this source was sufficient in the field until George Marsaglia came with the Diehard battery.

Diehard was developed at Florida State University by George Marsaglia in 1995 [46]. It was published on CD together with three so-called true random data sources and multiple *PRNGs*.

Crypt-X suite was a test battery for analysis of cryptoprimitives used in the late nineties [47]. NIST STS fully surpassed it.

NIST STS is the "golden" standard of statistical testing in cryptography [48]. Four tests of the battery were standardised as one step for the certification FIPS 140-2 [49]. It was also used as one criterion of evaluation of the AES candidates by Murphy in [50]. For example, Murphy found NIST STS failing for the AES candidate HPC, which detected possible weaknesses of the algorithm.

NIST STS was developed in the late nineties, and the last revision was released in 2010 [48]. The 2010 update removed a biased test *Lempel-Ziv Complexity of Sequences*. Further analysis of the bias in the p-values produced by NIST STS shows more dependent tests producing biased results. Overall NIST STS is accepted as the "golden" standard, so newer batteries extend it (at least the FIPS 140 subset of NIST STS battery) and relates to it.

Dieharder was developed by Robert G. Brown at the Duke University in 2004 [51]. It reimplements all tests of Diehard battery and extends it by multiple additional tests. Dieharder also contains three of the fifteen tests of the NIST STS battery. However, it suffers similar bias flaws as NIST STS, as detected Obrátil in the submitted thesis [52].

TestU01 is state of the art in statistical testing. It is a library, continuously developed by Université de Montréal initially published in 2007 [53]. The design of the tests is innovative in comparison to previous batteries. It's tests are parametrised to form more sub-tests. This approach outperforms all other statistical batteries, as will be shown in our results in Table 6.1. The TestU01 also implements multiple *PRNG* from different categories, observing the same results, which proves the non-randomness.

TestU01 implements several batteries for specific purposes. The batteries share some of the tests.

PractRand is another competitor to the TestU01 [54]. The author claims that the PracRand suits better for testing a large amount of data in comparison to other batteries. However, we have not testified this claim.

RaBiGeTe is a statistical battery for Windows users [55]. It implements some of the NIST STS's tests and some of Donald Knuth's tests.

There are also others statistical batteries: CryptoStat [56], YAARX [57], ENT [58], SPRNG [59], gjrand [60] and BSI's test suite [61].

6.1 Statistical batteries

We evaluated the dataset of well-known cryptographic functions with NIST STS (faster implementation by Sýs et al. was used [62]), Dieharder and TestU01 batteries. Those batteries were automated in the project *Randomness Testing Toolkit* (*RTT*) [63] as part of the master thesis of Ľubomír Obrátil [52]. *RTT* is an online service, which allows the user to upload binary data and it executes the batteries on them. It automatically detects the best possible setting of the batteries, as they are parametrised mainly by the size of the input file. After the execution, *RTT* postprocess the results and outputs a human readable interpretation of them. Similarly to the previous visualisations in this thesis, red highlight implies test rejected randomness hypothesis simplifying the table interpretation.

Such interpretation is straightforward. However, we show it may be incorrect in a very specific situation due to the statistical properties of the batteries. To prove that, we need to explain the interpretation in a statistical manner.

A single test assesses the data against the null hypothesis that the data are random. The test may reject the hypothesis implying the data seems non-random, or it may accept the hypothesis, where it means the test cannot find enough evidence for non-randomness of the data. However, the test may fail with an error of two types.

Type I error describes the acceptance of the null hypothesis when it is false (false positive). The test incorrectly states that the data are not random. This error occurs with the probability specified by the *significance level* α for random data.

Type II error describes the rejection of the null hypothesis when it is true (false negative). The test did not spot non-randomness of the data. The probability of type II error is also dependent on the α, and it is denoted as β.

The batteries consist of a set of tests, that are parametrised up to hundreds of sub-tests. If the *significance level* α is, for example, 1 %, then the probability that all 100 tests will correctly pass is $0.99^{100} = 36.6\,\%$ (suppose the tests are independent, which however do not hold for sub-tests). Therefore, we have to expect type I errors. The probability of type I error further increases if the tests are dependent. Sýs et al. [64] showed that some of the NIST STS tests are dependent and they proposed the interpretation of the battery based on calculated proportion of the failed tests. The same authors plan to revise also Dieharder battery, which contains similar ambiguity in interpretation. Part of these results is going to be published in Ľubomír Obrátil's thesis [52]. *RTT*'s interpretation takes into account the type I error.

6.1.1 Statistical batteries results

The comparison of multiple batteries can be seen in Table 6.1. The visualisation required changes of the table. The rows of the table represent cryptographic functions reduced to the given number of rounds. These functions with rounds may repeat if different statistical testing tools were able to detect non-randomness of the function reduced to a different number of rounds. A column represents a single statistical battery or EACirc tool with the guided local search metaheuristic. A cell contains the number of failed sub-tests of total sub-tests from the table header or EACirc's *rejection rate*. The red highlight denotes our interpretation, signalising the tool rejected randomness hypothesis.

Function$_{\textbf{rounds}}$	**NIST STS** (x/15)	**Dieharder** (x/27)	**Small Crush** (x/10)	**Crush** (x/32)	**Rabbit** (x/16)	**Alphabit** (x/4)	**Block Alphabit** (x/4)	**EACirc** (*rejection rate*)
AES_3	8	15	5	20	5	2	4	0.164
$BLAKE_1$	11	11	5	18	5	2	3	0.260
$Grain_2$	14	27	9/9	31/31	15	4	4	1.000
$Grain^*_6$	1	0	0	3	1	0	0	0.017
$Grøstl_2$	12	23	9	27	9	3	3	1.000
JH_6	12/13	27	10	30/31	15	4	4	1.000
$Keccak_2$	14	27	10	31	15	4	4	1.000
$Keccak_3$	0	1	1	11	4	0	3	0.017
$MD6_8$	9	19	5	16	8	2	3	0.995
$MD6^*_{10}$	0	0	0	2	3	0	0	0.016
$Rabbit_0$	1	1	0	4	3	1	1	0.017
$Rabbit^*_4$	0	0	0	3	3	1	1	0.009
$RC4^*_{\text{no rounds}}$	0	0	0	3	0	0	0	0.008
$Salsa20_2$	12	26	8	28	11	3	3	1.000
Single DES_4	7	22	7	26	11	4	4	0.496
Single DES_5	1	6	1	18	5	2	3	0.011
$Skein_3$	12	27	10	30	13	3	4	1.000
$Skein_4$	0	0	0	10	4	1	3	0.012
$SOSEMANUK_4$	13/13	27	10	31/31	16	4	4	1.000
TEA_4	8	19	6	15	4	2	3	0.729
TEA_5	0	3	2	4	1	0	3	0.015
Triple DES_2	12	26	9	31	15	3	4	1.000
Triple DES_3	1	4	1	4	0	1	1	0.012

Table 6.1: Application of the statistical batteries and EACirc on the data from round-reduced well-known cryptographic functions. The values in the table are described in Section 6.1.1. The results were computed by Randomness Testing Toolkit. Complete results with detailed information about which tests are passing can be found on RTT page as results from 2017-04-24 to 2017-04-28 [65].

From those results, we have multiple observations.

- EACirc performs similarly to NIST STS and TestU01 Small Crush. Dieharder is slightly better than EACirc.
- TestU01 Crush outperforms all other batteries having a disadvantage of the computation time and the amount of analysed data.
- The high number of failed test indicates the data are biased in multiple characteristics. All batteries usually reject such data. However, when only a single battery detects non-randomness, it is TestU01.
- **TestU01 Rabbit was able to detect deviances in full Rabbit cipher.** After analysis of this result, we found a report *On a bias of Rabbit* [66] during the eStream competition. The bias detected by TestU01 Rabbit may be the same. Even though this bias was found, Rabbit cipher was selected as software profile finalist, stating that the bias does not allow any feasible attack due to its complexity.
- A similar observation is the bias of a hardware profile eStream finalist Grain. TestU01 Crush can detect bias up to six-round Grain, which is almost one-half of the full function (13 rounds). The only known cryptanalysis of Grain is a *dynamic cube attack* [67]. It does not explain the bias observed by us. Our round reduction is based on a limitation of the shuffle of the non-linear feedback shift registers, which may require more than six rounds to shuffle the data enough. The full Grain cipher has 13 rounds, which still provides some security margin.
- The interpretation based on the probability of fail of n sub-tests would not detect the non-randomness property of five-round Grain. The rejection approach is strict (so it has a hight probability of type II error). However, a manual observation of the failing sub-tests are being the same as the failing sub-tests of four- and six-round Grain provides a necessary trust to our interpretation that the data are non-random. In addition, such results were examined via retesting on different data from the same setting.
- The results marked with an asterisk on TestU01 Crush and Rabbit were manually classified as non-random. We did not highlight them due to consistency, as the interpretation method leaves them as passed test. However, the probability of those extreme results is below 1 %.
- A rejection of randomness by the battery do not have to be as a black-box (as we present here), but we can analyse, which test failed. However, depending on the test, it may be more difficult than analysis of the output circuit from EACirc.

  EACirc may also provide concrete proof of the data dependency. If there is a dependency between specifics bytes, it would be present in the strong distinguisher's circuit. The statistical battery would fail some test, probably the *dependency test*. However, they would not state, what is the pattern in the data.
- Another advantage of EACirc is the simplicity of already learned distinguisher. Using learned distinguisher needs only kilobytes of the tested data. EACirc may even state the guess for a single *test vector*, such setting was used in [9]. The test of statistical battery always requires a fixed amount of data.

6.2 Compression algorithms

One of the definitions of randomness is via Kolmogorov's complexity. Data are random if they cannot be compressed. The data from the cryptographic function can always be compressed, but the computation would require finding the key used for the encryption. Therefore, such compression is infeasible.

We experimented with this approach, trying capabilities of compression programs. We have analysed following Unix lossless data compression utilities.

zip is compression utility based on Deflate algorithm. It combines LZ77 [68] and Huffman coding algorithms.

gzip is GNU version of *zip* utility.

lzop is compression utility using LZO algorithm. It aims for faster decompression; however, the compression can be less efficient than *gzip*.

bzip2 is compression algorithm based on Burrows–Wheeler transformation [69] of the reoccurring sequences to strings of identical letters. Then it applies the move-to-front transformation and Huffman coding.

NanoZip in version 0.09a is the latest version from the year 2011 of an experimental archiver software based on Burrows-Wheeler compression [70]. Based on multiple data compression benchmarks, NanoZip has the highest compression capabilities. Although we used various parameters, we did not obtain archive, which could be losslessly decompressed. Therefore, we omit this algorithm.

All the utilities were executed with the highest compression ratio with the slowest speed.

We denote the results in a similar manner as results of the batteries. The rows are labelled by cryptographic functions, and columns represent compression utilities. The cell contains the compression ratio, which says how much was saved by the compression. Compression of random data should not save anything, marked with a dash. Any higher value means the data are non-random. We tested this evaluation experimentally on random data, and the compressed file was never smaller than the original file. This property was tested for more than ten samples.

Observations from the results in Table 6.2 are following.

- The efficiency of newer *bzip2* on cryptographic data is not higher than the efficiency of *gzip* and *zip*. The fact that we have to reduce functions to very few rounds is probably caused by character (byte-level) specialisation of the compression algorithms. The cryptographic data may contain more bit-level dependencies.
- The artificial neural networks perform slightly better than compression algorithms. ANN have an advantage of bit-level data inspection, while they can test only a single *test vector* (16 bytes) of data.

6.3 Metaheuristics for randomness testing

Besides statistical batteries, numerous other works did the cryptanalysis based on testing some of the statistical properties. We have encountered several works using genetic algo-

Function$_{\text{rounds}}$	**bzip2**	**gz**	**lzop**	**zip**	**ANN**
AES_2	2 %	15 %	25 %	15 %	1.0
$BLAKE_0$	0 %	0 %	1 %	0 %	1.0
$Grain_2$	3 %	11 %	25 %	11 %	1.0
$Grøstl_0$	0 %	0 %	1 %	0 %	1.0
$HC\text{-}128_0$	–	–	–	–	0.497
JH_6	57 %	59 %	68 %	59 %	1.0
$Keccak_1$	16 %	99 %	36 %	99 %	1.0
$Keccak_2$	–	99 %	–	99 %	0.973
$MD6_6$	99 %	99 %	59 %	99 %	1.0
$MD6_7$	99 %	97 %	–	97 %	0.677
$Rabbit_0$	–	–	–	–	0.499
$RC4_{\text{no rounds}}$	–	–	–	–	0.504
$Salsa20_2$	–	99 %	–	99 %	0.771
Single DES_2	71 %	79 %	84 %	78 %	0.933
Single DES_3	–	–	–	–	0.637
$Skein_2$	93 %	79 %	93 %	80 %	0.999
$Skein_3$	–	–	–	–	0.577
$SOSEMANUK_4$	0 %	0 %	1 %	0 %	1.0
TEA_2	97 %	99 %	93 %	99 %	0.962
TEA_3	–	99 %	–	88 %	0.750
Triple DES_1	99 %	98 %	99 %	98 %	0.999
Triple DES_2	–	–	–	–	0.656

Table 6.2: Capabilities of compression algorithms in comparison with artificial neural networks on cryptographic data. Cells contain how much space was saved by the compression or the acceptance rate of the neural network.

rithms for statistical analysis of cryptoprimitives. Most of them analysed Tiny Encryption Algorithm (TEA), which is a simple block cipher designed by D. Wheeler and R. Needham in 1995 [71]. It was used as a benchmark due to its simplicity and well-known vulnerabilities [72]. The application of genetic algorithms started in 2002 by J. Hernández et al. [11], who found statistical deviance in two-round TEA. Further improvements from the team were presented in 2004 finding the non-randomness in three- and four-round TEA as well [12]. So far the best results using genetic algorithm were found by W. Hu [13], who detected non-random properties for five-round TEA. We compared EACirc with these results in [39].

Genetic algorithms were also used for analysis of four-round DES [14]. DES was also analysed using particle swarm metaheuristic in [73], and Simplified DES was analysed by the simulated annealing, and the tabu search in [74]. The so far best metaheuristic cryptanalysis on DES was on eight-round DES using genetic algorithm by Husein et al. [15]. Except for the last work, these methods are worse than the statistical batteries in the number of rounds, but they can provide additional information to the cryptanalyst that batteries do not.

7 Conclusion

7.1 Summary

This work analysed the application of metaheuristics in randomness testing tools, especially in EACirc. The possibility of their implementation with the potential issues was discussed in Chapter 3. We extended EACirc tool by three new metaheuristics that represented different single-solution metaheuristics types. Guided local search metaheuristic always performs better than the former EACirc's metaheuristic. In addition, we showed it significantly simplify output distinguisher, which reduces the complexity of further cryptanalysis. Therefore, guided local search will be incorporated into the following release of the EACirc tool as the default metaheuristic.

Another tested method was using the artificial neural network. We experimented with known models from pattern recognition field, without much success. As we examined the literature within this area, we have found no serious work on this problem. It is still an open question if the artificial neural networks can achieve better results in the cryptographic data recognition. To solve it out, more theoretical knowledge of ANN is necessary. The neural network algorithms may also need a significant redesign. The artificial neural network works with floating point number representation, and the discrete bit-level patterns do not fit the gradient descent algorithm for the selected data representation.

In comparison to state of the art in randomness testing, all our methods perform worse than the TestU01 statistical battery. In comparison with other statistical batteries, EACirc finally reached capabilities comparable with Dieharder. The only advantage of EACirc over TestU01 is the analysis of the developed distinguisher circuit, which may reveal concrete bytes dependencies. TestU01 Crush, the strongest battery setting, found bias of full Rabbit function, which probably replicated already known results discovered during eStream competition.

An important product of this work is the testbed of well-known cryptographic function, which is an extension and correction of previous testing sets. Besides, we presented *generator* tool for the direct generation of the tested data, which can be used by developers of others randomness testing tools as a benchmark. It can be used as well by cryptographers to generate specifically modified cryptography data, usually reduced in the number of rounds.

7.2 Future work

The capabilities of EACirc increased closer to the level of Dieharder battery. My estimate is that the current approach using byte-oriented circuits is unable to compete directly with the TestU01 battery. EACirc should focus more on the circuit analysis, which offers an added value over the TestU01.

Our team develop a bit-oriented randomness testing project. It does not utilise any metaheuristic yet; an application of one may increase its already promising capabilities. It fits well population-based metaheuristics, such as evolutionary algorithms or swarm intel-

ligence. The work on the integration of those metaheuristics already started within this thesis.

The testbed of well-known functions can be further extended by different means of reduction of the cryptographic functions RC4 and HC-128. For RC4, we can use bit selection capabilities of the *generator*. Other plans are to extend the tool by already implemented CAESAR functions and new *PRNGs*. We plan to publish the *generator* altogether with the *Randomness Testing Toolkit* for NordSec conference.

Presented implementation of neural networks may have limited results due to the analysis of a low amount of the tested data. A recurrent neural network, having an advantage of an internal state, may analyse a stream of data and try to predict the next bit of the input. Such approach would show if the limitation of the neural networks were the limited amount of observed data.

Regarding the artificial neural network, we began cooperation with pattern recognition team from University Ca' Foscari Venice. The results of this cooperation are still open.

Bibliography

[1] Tim Dierks. "The transport layer security (TLS) protocol version 1.2". In: (2008). URL: https://www.ietf.org/rfc/rfc5246.txt (visited on 05/20/2017).

[2] National Institute for Standards and Technology. *Announcing request for candidate algorithm nominations for the advances encryption standard (AES)*. 1997. URL: http://csrc.nist.gov/archive/aes/pre-round1/aes_9709.htm (visited on 05/20/2017).

[3] European Network of Excellence for Cryptology. *eStream project: Call for Stream Cipher Primitives*. 2005. URL: http://www.ecrypt.eu.org/stream/call/ (visited on 05/20/2017).

[4] National Institute for Standards and Technology. *SHA-3: Cryptographic hash algorithm competition*. 2007. URL: http://csrc.nist.gov/groups/ST/hash/sha-3/index.html (visited on 05/20/2017).

[5] CAESAR committee. *CAESAR: Competition for Authenticated Encryption: Security, Applicability, and Robustness*. 2013. URL: http://competitions.cr.yp.to/caesar-call.html (visited on 05/20/2017).

[6] Petr Švenda et al. *EACirc. Framework for automatic search for problem solving circuit via Evolutionary algorithms*. Centre for Research on Cryptography and Security, Masaryk University. 2012. URL: https://github.com/crocs-muni/eacirc (visited on 05/20/2017).

[7] David J. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. 3rd ed. CRC Press, 2003. ISBN: 978-1-58488-440-8.

[8] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons, 2009.

[9] Martin Ukrop. "Usage of evolvable circuit for statistical testing of randomness". Bachelor thesis. Faculty of Informatics Masaryk University, 2013. URL: https://is.muni.cz/th/374297/fi_b/ (visited on 05/20/2017).

[10] Marek Sýs and Petr Švenda and Martin Ukrop and Vashek Matyáš. "Constructing empirical tests of randomness". In: *Proceedings of the 11th International Conference on Security and Cryptography*. ICETE. IEEE. 2014, pp. 1–9. DOI: 10.5220/0005023902290237.

[11] Julio César Hernández et al. "Genetic Cryptoanalysis of Two Rounds TEA". In: *Computational Science—ICCS 2002*. Springer, 2002, pp. 1024–1031.

[12] Julio C Hernández and Pedro Isasi. "Finding Efficient Distinguishers for Cryptographic Mappings, with an Application to the Block Cipher TEA". In: *Computational Intelligence* 20.3 (2004), pp. 517–525.

[13] Wei Hu. "Cryptanalysis of TEA Using Quantum-Inspired Genetic Algorithms". In: *Journal of Software Engineering and Applications* 3.01 (2010), pp. 50–57. DOI: 10.4236/jsea.2010.31006.

[14] Jun Song et al. "Cryptanalysis of four-round DES based on genetic algorithm". In: *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*. IEEE. 2007, pp. 2326–2329.

[15] Hasan Mohammed Hasan Husein et al. "A Genetic Algorithm for Cryptanalysis of DES-8." In: *IJ Network Security* 5.2 (2007), pp. 213–219.

[16] Marco Dorigo and Luca Maria Gambardella. "Ant colony system: a cooperative learning approach to the traveling salesman problem". In: *IEEE Transactions on evolutionary computation* 1.1 (1997), pp. 53–66.

[17] Dominik Moritz and Matthias Springer. *Solving Satisfiability with Ant Colony Optimization and Genetic Algorithms*. 2010.

[18] Donald Olding Hebb. *The Organization of Behavior*. New York: John Wiley & Sons, 1949.

[19] Paul Werbos. "Beyond regression : new tools for prediction and analysis in the behavioral sciences". Ph. D. thesis. Harvard University, 1975. URL: https://www.researchgate.net/publication/35657389_Beyond_regression_new_tools_for_prediction_and_analysis_in_the_behavioral_sciences (visited on 05/20/2017).

[20] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org (visited on 05/20/2017).

[21] Li-Hua Li, Luon-Chang Lin, and Min-Shiang Hwang. "A remote password authentication scheme for multiserver architecture using neural networks". In: *IEEE Transactions on Neural Networks* 12.6 (2001), pp. 1498–1504.

[22] Martín Abadi and David G Andersen. "Learning to Protect Communications with Adversarial Neural Cryptography". In: *arXiv preprint arXiv:1610.06918* (2016).

[23] Khalil Shihab. "A backpropagation neural network for computer network security". In: *Journal of Computer Science* 2.9 (2006), pp. 710–715.

[24] Eva Volna et al. "Cryptography Based On Neural Network." In: *ECMS*. 2012, pp. 386–391.

[25] Ľubomír Obrátil. *Randomness Testing Toolkit. Results of statistical testing batteries on Quantis data*. Centre for Research on Cryptography and Security, Masaryk University. 2017. URL: http://rtt.ics.muni.cz/ViewResults/Experiment/1732/ (visited on 05/20/2017).

[26] M.E. O'Neill. *PCG, A Family of Better Random Number Generators. PCG is a simple and fast statistically good PRNG*. 2015. URL: http://www.pcg-random.org/ (visited on 05/20/2017).

[27] Matej Prišťák. "Automatické hledání závislostí u proudových šifer projektu eStream". Master thesis. Faculty of Informatics Masaryk University, 2012. URL: http://is.muni.cz/th/324866/fi_b_a2/ (visited on 05/20/2017).

[28] Ondrej Dubovec. "Automatické hledání závislostí u kandidátních hašovacích funkcí SHA-3". Bachelor thesis. Faculty of Informatics Masaryk University, 2012. URL: http://is.muni.cz/th/172546/fi_m/ (visited on 05/20/2017).

[29] Martin Ukrop. "Randomness analysis in authenticated encryption systems". Master thesis. Faculty of Informatics Masaryk University, 2016. URL: https://is.muni.cz/th/374297/fi_m/ (visited on 05/20/2017).

[30] Makoto Matsumoto and Takuji Nishimura. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 8.1 (1998), pp. 3–30.

[31] Karel Kubíček et al. *EACirc – documentation wiki. Streams for data manipulation*. Centre for Research on Cryptography and Security, Masaryk University. 2016. URL: https://github.com/crocs-muni/eacirc/wiki/Streams (visited on 05/20/2017).

[32] Brad Conte. *crypto-algorithms. Basic implementations of standard cryptography algorithms*. 2012. URL: https://github.com/B-Con/crypto-algorithms (visited on 05/20/2017).

[33] Wikipedia. *Tiny Encryption Algorithm. Wikipedia, The Free Encyclopedia*. 2013. URL: https://en.wikipedia.org/w/index.php?title=Tiny_Encryption_Algorithm&oldid=748477062 (visited on 05/20/2017).

[34] Petr Švenda et al. *EACirc Streams. A library and tool for data generation of specifically reduced cryptographic function*. Centre for Research on Cryptography and Security, Masaryk University. 2017. URL: https://github.com/crocs-muni/eacirc-streams (visited on 05/20/2017).

[35] Petr Švenda and Martin Ukrop and Vashek Matyáš. "Towards cryptographic function distinguishers with evolutionary circuits". In: *Security and Cryptography (SECRYPT), 2013 International Conference on*. ICETE. IEEE. 2013, pp. 135–146. DOI: 10.5220/0004524001350146.

[36] Ľubomír Obrátil. "Automated task management for BOINC infrastructure and EACirc project". Bachelor thesis. Faculty of Informatics Masaryk University, 2015. URL: https://is.muni.cz/th/410282/fi_b/ (visited on 05/20/2017).

[37] Karel Kubíček et al. *EACirc-utils. Utilities for EACirc framework*. Centre for Research on Cryptography and Security, Masaryk University. 2012. URL: https://github.com/crocs-muni/eacirc-utils (visited on 05/20/2017).

[38] Marek Sýs et al. *EACirc – lookup table (LUT) cipher – discontinued. Generator with slow increase of entropy for benchmarking randomness testing*. Centre for Research on Cryptography and Security, Masaryk University. 2016. URL: https://github.com/crocs-muni/eacirc/commit/9808dda2401c15fac6f4984ba308140ebd2e3162 (visited on 05/20/2017).

[39] Karel Kubíček and Jiří Novotný and Petr Švenda and Martin Ukrop. "New results on reduced-round Tiny Encryption Algorithm using genetic programming". In: *INFOCOMMUNICATIONS JOURNAL* 8.1 (2016). URL: http://crocs.fi.muni.cz/_media/public/papers/infocom/infocommunications2016.pdf.

[40] Karel Kubíček et al. *EACirc – documentation wiki. Known bugs*. Centre for Research on Cryptography and Security, Masaryk University. 2016. URL: https://github.com/crocs-muni/eacirc/wiki/Known-bugs (visited on 05/20/2017).

[41] John Ellson et al. "Graphviz—open source graph drawing tools". In: *International Symposium on Graph Drawing*. Springer. 2001, pp. 483–484.

[42] François Chollet et al. *Keras: Deep learning library for Theano and Tensorflow*. 2015. URL: https://keras.io/ (visited on 05/20/2017).

[43] Martín Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016). URL: https://www.tensorflow.org/ (visited on 05/20/2017).

[44] Karel Kubíček. *EANet. An artificial neural network version of EACirc tool*. Centre for Research on Cryptography and Security, Masaryk University. 2017. URL: https://github.com/crocs-muni/eacirc-streams (visited on 05/20/2017).

[45] Donald Ervin Knuth. *The Art of Computer Programming. Seminumerical Algorithms*. First. Vol. 2. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1969. ISBN: 9780201038026.

[46] George Marsaglia. *Diehard: a battery of tests of randomness*. Floridan State University. 1995. URL: https://web.archive.org/web/20040810115625/http://stat.fsu.edu/~geo/diehard.html (visited on 05/20/2017).

[47] William Caelli et al. *Crypt-X suite. Randomness testing suite*. 1998. URL: https://web.archive.org/web/19990224063612/http://www.isrc.qut.edu.au/cryptx/index.html (visited on 05/20/2017).

[48] Andrew Rukhin et al. *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. Tech. rep. DTIC Document, 2001. URL: http://csrc.nist.gov/groups/ST/toolkit/rng/documents/SP800-22rev1a.pdf (visited on 05/20/2017).

[49] PUB FIPS. "FIPS 140-2". In: *Security Requirements for Cryptographic Modules* 25 (2001).

[50] Sean Murphy. "The power of NIST's statistical testing of AES candidates". In: *Preprint. January* 17 (2000). URL: http://csrc.nist.gov/archive/aes/round2/conf3/papers/AES3papers-2.zip (visited on 05/20/2017).

[51] Robert G Brown, Dirk Eddelbuettel, and David Bauer. "Dieharder: A random number test suite". Version 3.31.1. In: *Open Source software library, under development* (2013). URL: http://www.phy.duke.edu/~rgb/General/dieharder.php (visited on 05/20/2017).

[52] Ľubomír Obrátil. "The automated testing of randomness with multiple statistical batteries". Master thesis. Faculty of Informatics Masaryk University, submitted in 2017. URL: https://is.muni.cz/th/410282/fi_m/. Forthcoming.

[53] Pierre L'Ecuyer and Richard Simard. "TestU01: A C Library for Empirical Testing of Random Number Generators". In: *ACM Transactions on Mathematical Software (TOMS)* 33.4 (Aug. 2007). ISSN: 0098-3500. DOI: 10.1145/1268776.1268777.

[54] Chris Doty-Humphrey. *Practically Random: Specific tests in PractRand*. 2014. URL: http://pracrand.sourceforge.net/ (visited on 05/20/2017).

[55] Cristiano Piras. *RaBiGeTe Documentation. Random Bit Generators Tester*. 2004. URL: http://cristianopi.altervista.org/RaBiGeTe_MT/ (visited on 05/20/2017).

[56] Alan Kaminsky and Jessica Sorrell. "CryptoStat: a Bayesian Statistical Testing Framework for Block Ciphers and MACs". In: *Rochester Institute of Technology, Rochester, NY* (2013). URL: http://www.cs.rit.edu/~ark/students/jls6190/report.pdf (visited on 05/20/2017).

[57] Alex Biryukov and Vesselin Velichkov. "Automatic search for differential trails in ARX ciphers". In: *Cryptographers' Track at the RSA Conference*. Springer. 2014, pp. 227–250. URL: https://publications.uni.lu/bitstream/10993/17183/1/automatic-search-arx-trails.pdf (visited on 05/20/2017).

[58] John Walker. *Ent: A Pseudorandom Number Sequence Test Program*. Fourmilab. 2008. URL: https://www.fourmilab.ch/random/ (visited on 05/20/2017).

[59] Michael Mascagni and Ashok Srinivasan. "Algorithm 806: SPRNG: A scalable library for pseudorandom number generation". In: *ACM Transactions on Mathematical Software (TOMS)* 26.3 (2000), pp. 436–461.

[60] Geronimo Jones. *gjrand random numbers*. 2007. URL: http://gjrand.sourceforge.net/ (visited on 05/20/2017).

[61] Werner Schindler and Wolfgang Killmann. "Evaluation criteria for true (physical) random number generators used in cryptographic applications". In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer. 2002, pp. 431–449.

[62] Marek Sýs, Zdeněk Říha, and Vashek Matyáš. "Algorithm 970: Optimizing the NIST Statistical Test Suite and the Berlekamp-Massey Algorithm". In: *ACM Transactions on Mathematical Software (TOMS)* 43.3 (2016), p. 27.

[63] Ľubomír Obrátil. *Randomness Testing Toolkit. Randomness testing toolkit automates running and evaluating statistical testing batteries*. Centre for Research on Cryptography and Security, Masaryk University. 2016. URL: http://rtt.ics.muni.cz/ (visited on 05/20/2017).

[64] Marek Sýs et al. "On the interpretation of results from the NIST statistical test suite". In: *Romanian Journal of Information Science and Technology* 18.1 (2015), pp. 18–32.

[65] Ľubomír Obrátil. *Randomness Testing Toolkit. Results of statistical testing batteries on dataset of well-known cryptographic functions*. Centre for Research on Cryptography and Security, Masaryk University. 2017. URL: http://rtt.ics.muni.cz/ViewResults/?created_from=2017-04-24+12:00:00&created_to=2017-04-29+00:00:00 (visited on 05/20/2017).

[66] Jean-Philippe Aumasson. "On a bias of Rabbit". In: *State of the Art of Stream Ciphers Workshop (SASC 2007), eSTREAM, ECRYPT Stream Cipher Project, Report*. 2007.

[67] Itai Dinur and Adi Shamir. "Breaking Grain-128 with dynamic cube attacks". In: *International Workshop on Fast Software Encryption*. Springer. 2011, pp. 167–187.

[68] Jacob Ziv and Abraham Lempel. "A universal algorithm for sequential data compression". In: *IEEE Transactions on information theory* 23.3 (1977), pp. 337–343.

[69] Michael Burrows and David J Wheeler. "A block-sorting lossless data compression algorithm". In: (1994).

[70] Sami Runsas. *NanoZip. Experimental file archiver software*. 2014. URL: http://nanozip.ijat.my/ (visited on 05/20/2017).

[71] David J Wheeler and Roger M Needham. "TEA, a tiny encryption algorithm". In: *Fast Software Encryption*. Springer. 1995, pp. 363–366.

[72] John Kelsey, Bruce Schneier, and David Wagner. "Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X, NewDES, RC2, and TEA". In: ICICS '97 (1997), pp. 233–246. URL: http://dl.acm.org/citation.cfm?id=646277.687180 (visited on 05/20/2017).

[73] Waseem Shahzad, Abdul Basit Siddiqui, and Farrukh Aslam Khan. "Cryptanalysis of four-rounded DES using binary particleswarm optimization". In: *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*. ACM. 2009, pp. 2161–2166.

[74] N Nalini and G Raghavendra Rao. "Cryptanalysis of simplified data encryption standard via optimization heuristics". In: *Intelligent Sensing and Information Processing, 2005. ICISIP 2005. Third International Conference on*. IEEE. 2005, pp. 74–79.

A Glossary

Activation function of artificial neural network is processing the sum of inputs of the neurone to the output. It restricts the output to given interval, usually to $(-1, 1)$.

Benchmarking functions are a subset of the well-known cryptographic functions analysed in this thesis, such that are possible, yet not straightforward for EACirc detection. Therefore, some of the runs can find significant distinguisher, while others do not. Therefore, such combination of function and round are useful for comparison of metaheuristics.

Binary weight denotes the number of ones in the data.

Connection bug is a statistical issue of EACirc's evaluation. It is caused by evaluation of circuits, which has not connected the input to the output. Therefore, the circuit produces the same distribution of the output byte per dataset. These distributions are compared by χ^2 test, which fails with 0 for having not enough categories.

Convoluted neural network (CNN) is a type of artificial neural network, where the first few layers are not entirely connected, but they form smaller networks for detection of local patterns.

Critical value corresponds to the *significance level*. The *significance level* is the volume of distribution's extremes; the *critical value* is the p-value of such volume.

Empirical critical value is a minimal measured *rejection rate*, after which we reject the randomness hypothesis. Its value would be theoretically $2 \times \alpha$ for 1 000 EACirc runs. However, due to the *connection bug* is the actual value slightly higher.

Epoch is the number of iterations evaluated on the same dataset. After the end of each *epoch*, the dataset is regenerated.

Fitness function is a method for comparison of individual quality. The *fitness function* is in this work defined for maximisation problem. Therefore, *fitness* equal to zero is the worst, while *fitness* equal to one is the best.

Generator is a tool build upon implementation of cryptographic functions from EACirc allowing generation of round-reduced and further customised cryptographic data.

Local optimum of the *fitness* landscape are local maxima. As hill-climbing methods get trapped in such solution, various optimisation techniques were developed, usually with the goal evading the *local optima*.

Monobit test is the basic statistic test. It tests the frequency of ones and zeroes in the data. It expects to have a similar amount of ones and zeroes.

Neighbour solution is a candidate solution created by a single change by the *mutation* operation. For example, in EACirc it is a single addition of removal of a connector or change of the Boolean operator in the node.

Neighbourhood is the solution space around the current individual.

Overfitting is a situation of the learning process when the individual specialises too tightly to the learning data. Therefore, we change the learning datasets each *epoch*,

so the learning process leads to generalisation instead of the individual being too accurate.

Pseudo-random number generator (PRNG) is a deterministic generator of pseudo-random numbers, fully depending on its initial seed.

Pruning is the process of removal unused connectors and nodes from the circuit. *Pruning* is done by breath first search from the output with prior knowledge, what is the arity of functions in nodes.

Quantum random number generator (QRNG) is an instance of a truly random number generators, where the physical process corresponds to a measurement of a quantum event, such as photon's polarisation.

Randomness Testing Toolkit (RTT) is framework providing a unified interface to statistical batteries NIST STS, Dieharded and TestU01.

Round-reduced cryptographic function is a cryptographic function, where the iterative part such as Feistel network is reduced to given number of rounds. Zero rounds mean the iterative part is skipped at all. However, even zero rounds function can produce randomly looking data, as it may consist of strong pre-computation and post-computation. Some functions are irreducible.

Rejection rate is the number of EACirc runs that rejected the randomness hypothesis divided by a total number of runs, usually 1 000 runs.

Significance level α defines the probability that test rejects the null hypothesis when it is true. The *significance level* can usually be parametrised.

Solution exploitation is the process of advancement of an already feasible solution.

Space exploration is the process of broad search in the solution space for a feasible solution.

Strict avalanche criterion (SAC) says that a single bit flip in the input of the hash function should lead to flipping roughly half of the output bits. Even though it was stated for hash functions, it should apply to cryptographic ciphers as well.

Tabu list is a set of visited individuals. Tabu search metaheuristic would not visit these solutions again to prevent repeating computation.

Test vector is the least tested unit of data. It is usually a single ciphertext.

Trully-random number generator (TRNG) (sometimes called hardware RNG) generates the random data from the physical process.

Weight evaluator is a method for *fitness* evaluation based on the *binary weight* of the outputs of the circuit. If the number of ones output by the circuit per tested dataset significantly differs from the number of ones from the reference dataset, then the tested data are non-random.

B Data attachment

The data attachment available in the thesis repository[1] contains source codes, dumps of the results, and the configuration for their replication. The files have following structure.

- `eacirc`

  The source code of EACirc's branch metaheuristics with commit `21fd6fa` from 2017-04-18. The repository contains as git submodule also the source of the *generator*, in the directory `eacirc-streams` on the dev branch with commit `15c1ad1` from 2017-04-18.
- `eacirc.wiki`

  Project's wiki-based documentation, with commit `df6e270` from 2017-05-08.
- `eacirc-streams.wiki`

  Generator's wiki-based documentation, with commit `bb51a84` from 2017-05-05.
- `eacirc-utils`

  The source code of EACirc's utilities with master commit `91664f5` from 2017-04-25.
- `eanet`

  The source code of the experiments with artificial neural networks with commit `aedc153` from 2017-05-20.
- `results`

  Underlying data for the thesis experiments. Contains folders `related_work` with results of the compression and batteries, `ann` with results of the artificial neural networks together with the visualisation of the learning process, and `ss_metaheur` with results, binaries, and replication scripts of single-solution metaheuristics. The last folder also contains the analysis of the circuits.
- `thesis-src`

  Thesis text source including bibliography and used images.

1. `http://is.muni.cz/th/408351/fi_m/`

C Single-solution metaheuristics results

Function\rounds	**-1**	**0**	**1**	**2**
$rnd\_rnd_{\text{no rounds}}$	–	0.01681	–	–
AES_3	1.0	0.305	0.026	–
$BLAKE_1$	1.0	0.051	0.018	0.021
$Grain_2$	–	1.0	0.023	0.018
$Grøstl_2$	–	1.0	0.014	0.016
$HC\text{-}128_{\text{no rounds}}$	–	0.015	–	–
JH_6	–	1.0	0.016	0.018
$Keccak_3$	1.0	0.012	0.026	–
$MD6_8$	–	0.419	0.016	0.018
$Rabbit_0$	–	0.019	0.025	0.019
$RC4_{\text{no rounds}}$	–	0.02	–	–
$Salsa20_2$	–	1.0	0.019	0.019
$SINGLE\text{-}DES_4$	1.0	0.093	0.017	0.02
$Skein_3$	1.0	1.0	0.025	0.015
$SOSEMANUK_4$	–	1.0	0.019	0.025
TEA_4	1.0	0.244	0.015	0.024
Triple DES_2	–	1.0	0.015	0.029

Table C.1: Results of simulated annealing as EACirc metaheuristic on the well-known cryptographic functions.

Function\rounds	-1	0	1	2
rnd_rnd$_{\text{no rounds}}$	–	0.01191	–	–
AES_3	1.0	0.164	0.017	–
$BLAKE_1$	1.0	0.260	0.017	0.013
$Grain_2$	–	1.0	0.011	0.009
$Grøstl_2$	–	1.0	0.011	0.017
HC-128$_{\text{no rounds}}$	–	0.011	–	–
JH_6	–	1.0	0.010	0.010
$Keccak_3$	1.0	0.017	0.013	–
$MD6_8$	–	0.995	0.013	0.016
$Rabbit_0$	–	0.007	0.003	0.013
RC4$_{\text{no rounds}}$	–	0.008	–	–
$Salsa20_2$	–	1.0	0.016	0.012
Single DES_4	1.0	0.496	0.011	0.013
$Skein_3$	1.0	1.0	0.012	0.006
$SOSEMANUK_4$	–	1.0	0.015	0.009
TEA_4	1.0	0.729	0.015	0.006
Triple DES_2	–	1.0	0.012	0.007

Table C.2: Results of guided local search as EACirc metaheuristic on the well-known cryptographic functions.

Function\rounds	-1	0	1	2
rnd_rnd$_{\text{no rounds}}$	–	0.01124	–	–
AES_3	1.0	0.182	0.015	–
$BLAKE_1$	1.0	0.157	0.016	0.012
$Grain_2$	–	1.0	0.012	0.002
$Grøstl_2$	–	1.0	0.016	0.008
HC-128$_{\text{no rounds}}$	–	0.009	–	–
JH_6	–	1.0	0.010	0.012
$Keccak_3$	1.0	0.014	0.011	–
$MD6_8$	–	0.992	0.010	0.017
$Rabbit_0$	–	0.010	0.014	0.018
RC4$_{\text{no rounds}}$	–	0.005	–	–
$Salsa20_2$	–	1.0	0.015	0.008
Single DES_4	1.0	0.441	0.009	0.013
$Skein_3$	1.0	1.0	0.011	0.009
$SOSEMANUK_4$	–	1.0	0.007	0.009
TEA_4	1.0	0.877	0.016	0.010
Triple DES_2	–	1.0	0.009	0.004

Table C.3: Results of variable neighbourhood search as EACirc metaheuristic on the well-known cryptographic functions.