An Online Platform for Testing and Evaluating Random Number Generators

Pawel Kubczak, Wiktor Wozniak, Jakub Nikonowicz, Lukasz Matuszewski, Mieczyslaw Jessa
The Faculty of Computing and Telecommunications
Poznan University of Technology
Poznan, Poland
{pawel.kubczak, jakub.nikonowicz, lukasz.matuszewski, mieczyslaw.jessa}@put.poznan.pl,
wiktor.ma.wozniak@doctorate.put.poznan.pl

***Abstract*—This article introduces a new online platform for testing binary random number generators. The growing share of low-complex devices in IoT networks increases the demand for basic authorization and authentication tools, the critical block of which is a secure random number generator. Communication devices, therefore, require designers to carry out time-consuming tests and acquire specialist knowledge of statistical testing in evaluation of their results. To meet the current requirements, we have created a test platform to assess the quality of random strings produced by the generator. The presented solution, based on the proprietary evaluation metric, provides feedback on the properties of the uploaded random sequences. Clear interface provides ease of use and by machine learning in the platform's backend, along with the increase of processed data, the improved quality of the interpretation delivered by the system is ensured. The operation of the platform has been confirmed experimentally, based on the analysis of hardware generators producing random strings with known properties.**

***Index Terms*—random number generator, pattern recognition, sequence similarity, statistical test**

I. Introduction

The rapid growth of wireless networks driven by the development of the Internet of Things (IoT) has a significant impact on the development of modern communication technologies, as well as related fields. The continuous increase in the number of devices that perform wireless communication escalates the demand for basic security mechanisms to ensure reliable authentication and authorization. In the required security systems, the core block is a random number generator (RNG), the non-deterministic nature of which determines the provided security level [1]. The production of even low-complex wireless network devices requires the designer to take into account the RNG in the designed structure. Since there are many publicly available structures, often based on ring oscillators (RO) [2], no specialist knowledge of using nondeterministic effects and harvesting entropy is required. However, the quality of the generated random sequences, especially in simple solutions such as RO, remains strongly dependent on the production parameters of the hardware platform and operating conditions [3]. Therefore, the specified generator design requires an iterative testing and improvement process. Random string testing is a well-established field in which the National Institute of Standards and Technology (NIST) plays a leading role. This organization has issued a special publications SP 800-22 [4] and SP 800-90B [5] which outline the requirements and methods for testing random sequences. Alternatives are the AIS-31 standard on Evaluation Methodology for Physical Random Number Generators, defined by the German Federal Office for Information Security (BSI) [6] and the TestU01 [7]. In the above cases, a set of publicly available, predefined tests helps in determining whether the generator produces independent and identical samples. Beyond the statistical test packages provided by government agencies, the frequently and willingly used DieHarder a random number test suite by R. Brown [8] should also be mentioned. The use of the above tests causes several inconveniences for potential designers and testers. First and foremost, the above-mentioned test suites require a detailed study of the technical documentation to resolve what statistical properties a given test is examining. Moreover, each test, apart from the binary decision, returns a result that requires independent interpretation, e.g. p-value. This forces the user to possess specialist knowledge in the field of statistical tests and the sources of their disturbances. Finally, tests are provided as precompiled code that requires a large amount of input data and provision of own computational hardware. Therefore, obtaining a low-data demanding yet effective procedure that evaluates random string against a single reference and returns clear information about the sequence quality remains an open issue.

To meet the growing demand for intelligible and approachable testing of RNGs, in this article we present an Online Platform for Testing and Evaluating Random Number Generators. The architecture of the prepared system is based on the original random sequence matching algorithm, performing feature extraction and returning an easy-to-use generator evaluation metric [9]. The use of machine learning in the backend ensures a constantly improving ready-made interpretation of the results, delivered to the user in an accessible form by the frontend. Performing an online test also ensures that the hardware load is transferred to the testing server. After uploading a sequence of random numbers from developed generator, the user receives direct feedback on the designated metrics and their prospective interpretation. As a result, the

This work was supported by the Polish Ministry of Science and Higher Education under the status activity task 0314/SBAD/0203 in 2021.

requirements of the iterative process of testing and improving the generator, both in terms of knowledge and necessary equipment, are significantly reduced.

The prepared online platform was pre-trained based on random number generators with known properties. Its operation has been confirmed experimentally in the evaluation of RO-based hardware generators implemented in FPGAs from leading manufacturers.

The rest of the article is organized as follows. Section II describes features of the proposed system. Section III presents the implementation, explains the experiment methodology and shows numerical results. Finally, Section IV gives the concluding remarks.

II. Proposed system

The core of the designed system is the original random sequence alignment algorithm that returns a bivalent metric. However, ensuring the targeted functionality requires the automation of data processing. For this purpose, the algorithm was used as a feature extraction function that provides data for machine learning embedded in the developed system.

A. Feature extraction

Motivated by the desired functionality of random sequence testing and inspired by the narrow group of sequence alignment methods used in arranging molecular sequences, we have created an innovative algorithm for comparative analysis. The algorithm performs mutual alignment of random series by a successive search for pairs of matching elements. An important feature of the algorithm is the careful introduction of sequence breaks for elements without match.

The algorithm starts with the fetch of two sequences S_1 and S_2. The trailing index k successively indicates the elements of the compare instruction, i.e., $S_1[k] == S_2[k]$. If the condition is true, the k simply increases to the next pair of elements. In the case of a mismatched pair in k-th position, i.e., $S_1[k] \neq S_2[k]$, the algorithm introduces an auxiliary index $p = k + 1$. Match optimization is performed by additional check that any element of S_1 matches any element of S_2 in the current range defined by distancing indexes k and p. Thus, the algorithm compares $S_{2,1}[a]$ to the current reference $S_{1,2}[p]$, where $a \in \langle k, p \rangle$. Failure to match within the running range results in an increase of p by one and jump to the new reference $S_{1,2}[p]$. The element found with the lowest possible index a preceding current p corresponds to the matching optimization, thus aborts the search and triggers the gapping for mismatched elements from both sequences. An organized search enhances the breaking of sequences evenly, rather than just one of them. An exemplary operation of the algorithm is shown in Figure 1 and detailed step-by-step description is provided in [9].

As a result, both sequences are stretched into the interleaving of gaps and matches that is subject to the target evaluation metric. The algorithm returns a two-element metric $S(a, b)$. The first is a stretch ratio determined as the ratio of the length of the gapped sequences $L_1^{'}$ and $L_2^{'}$ to their initial lengths L_1 and L_2. However, as the alignment progresses, the sequences dynamically expand. Consequently, in one sequence a "tail" of length L_t may occur, where L_t is the number of elements for which there were not enough elements to match in the second sequence. Therefore, the stretch metric a is calculated as follows

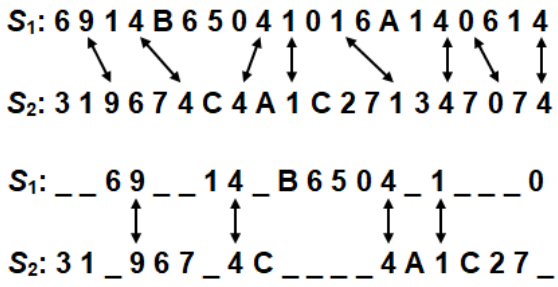


Fig. 1. An example of two random sequences with common elements (top) adjusted by the algorithm (bottom).

$$a = \frac{L_1^{'} + L_2^{'} - L_t}{L_1 + L_2 - L_t} \tag{1}$$

The second metric, b, is the cost of stretching the sequences. As support to distinguish between scattered and clustered gaps, cost is determined as an exponential function

$$b = \frac{\sum_i \sum_j 2^{G_i^j - 1}}{L_1 + L_2 - L_t} \tag{2}$$

where G_i^j is equal to the number of gaps in the j-th gap cluster in the i-th sequence.

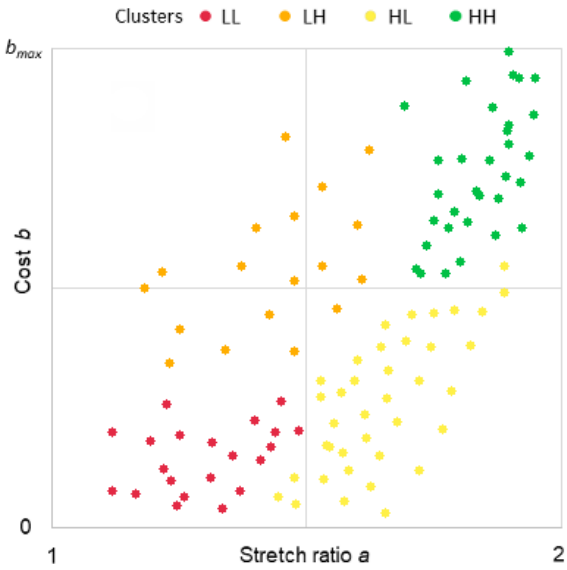


Fig. 2. The meaning groups of the $S(a, b)$ metric in the plane of stretch and cost.

The result interpreted on the (a, b)-plane allows to infer about statistical properties of sequences, thus to evaluate their source. The plane indicates four information regions (Fig. 2):

- LL – low stretch ratio and low cost indicate scattered and short mismatches, therefore high similarity of sequences. The classification indicates a low quality of the tested strings.
- LH – low stretch ratio with high cost characterizes insertion of gaps in series, i.e. cyclical convergences and divergences of both sequences. The assignment recognizes problems with maintaining the quality of sequences.
- HL – high stretch ratio with low cost indicates numerous scattered gaps, i.e., short matches and mismatches. The result is a premise for inference about the statistical similarity of the sequences.
- HH – high stretch ratio and high cost are the results of clustering gaps and sparse matches, thus negligible similarity of both sequences. Such classification confirms the high quality of the examined strings.

B. Machine learning

Dividing a multidimensional space into four sets of points representing the respective interpretation groups leads to a complex problem that is often difficult to describe mathematically in a dynamic environment. While obtaining sufficient theoretical models can be expensive and time-consuming, machine learning provides a heuristic tool for solving complex problems when deterministic approaches are overly complex or have poor performance. In the considered issue it can be efficiently used to find the optimal demarcation function. Machine learning falls into two categories, that are unsupervised learning (UL) and supervised learning (SL) [10]–[12]. The first approach learns from the input data that has not been labeled. Instead of responding to known feedback, unsupervised learning identifies commonalities in the data by similarity metric evaluated by internal compactness of the created clusters. The latter is used to build a data model based on inputs assigned to the pre-classified output. Through iterative optimization of the objective function, supervised algorithms learn a function for classification when the result is limited to a discrete set of values.

Given the interactive nature of the created platform, in the proposed solution we use the so-called semi-supervised learning (SSL) that combines both of the above categories. The applied SSL method is based on the cluster assumption, i.e., that the points of each class tend to form a cluster. Following, points that are in the same group are likely to belong to the same class. This assumption justifies the very existence of classes, i.e., if there is a densely populated continuum of objects it may seem unlikely that it will be divided into different classes [10], [12]. Under this assumption, unlabeled data helps to find the boundaries of each cluster more accurately – running the clustering algorithm allows obtaining pre-grouped points for a class assignment to each cluster. The resulting set of classified data creates an input for supervised learning that is used to develop a predictive model for new data flowing into the system. In this application, SSL utilizes unlabeled data to construct a classifier whose results exceed those obtained using only a small prelabeled data set [11]. Moreover, in the described configuration, the classification algorithm created from pre-existing data may be cyclically adjusted as future data become available to increase accuracy.

C. System architecture

The system architecture overview is shown in Figure 3. The operation of the system can be divided into two cycles. The request-response cycle is responsible for handling user requests. The user uploads the data into the system. Both random sequences are stored in the database and passed to the feature extraction algorithm. The algorithm returns the determined bivalent metric, which is added to the sequence record in the database and feed to the classifying function. As a response of the system, the user receives directly the calculated metric, its visualization on the plot in relation to the other metrics stored in the database, and analysis in accordance with the class assigned by the classifier.

SSL cycle implemented in the system backend is responsible for the improvement of the classifier accuracy. The update is triggered after each 10% increment in the number of entries in the database. Based on the stored metrics, the UL algorithm performs re-clustering. Re-labeled database entries are used as training data in SL, which returns updated demarcation functions used in the classifier.

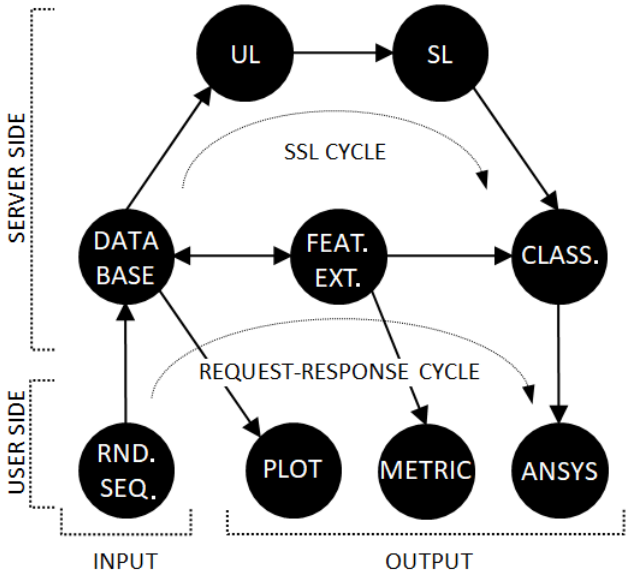


Fig. 3. Graph of the system data flow in the processing of the user's request.

III. Implementation and Experiment

The presented idea is implemented as a web application accessible at the URL address www.orangutan.edu.pl using any web browser. The solution is designed in a client-server model and implemented using Python 3.7 language combined with open-source Django framework version 3.2 [14]. Application's name *ORANGUTAN* is an acronym of the Online platform for RAndom Number Generator UTility ANalysis.

It is hosted employing a Linux server installed at Poznan University of Technology and is a free web tool for analyzing binary sequences.

A. User side

The data presentation layer, the so-called front-end, is based on Model View Template (MVT) architecture. The Model reflects the application logic and defines the format of the stored data. The View describes how to interpret a certain part of a model within a user interface. The Template defines how the data specified by the view is presented [13].

The home page of the ORANGUTAN application is shown in Figure 4. This page as well as all subpages are rendered using the layout designed using Cascading Style Sheets (CSS) with the Bootstrap 4.6 elements embedded [15]. The layout's top bar consists of the navigation bar, the logo of the Poznan University of Technology, and the application's logo. Subpages can be accessed through *Navbar* navigation links: *Home*, *About*, *Team* and *Contact*. The *About* page contains a description of the whole project including references to the related papers, e.g., detailing the back-end algorithms of the system. The *Team* page contains brief biographies of team members and the *Contact* page provides address data and a simple Google map with the Institute pointed.

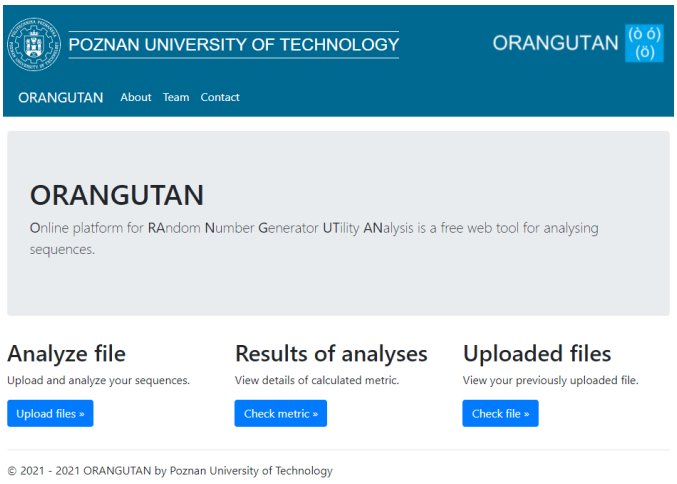


Fig. 4. The layout of the Orangutan's home page.

Directly from the ORANGUTAN's home page, the three subpages related to files and metrics can be accessed:

- Upload files – a form shown in Figure 5, designed for uploading binary files containing random sequences. Files have to be in *.bin* extension and size can not exceed 200 MB. It is mandatory to attach both files. Field with email address also has to be fulfilled. Description fields are optional and make the uploaded files easily recognizable. The form is secured against robots using the packet Django-ReCaptcha 2.0.6 provided by Google. Successful submission of the form triggers automatic generation of the metrics followed by the view with uploaded files and metrics details. Additionally, an e-mail message is sent to the provided address.
- Check metrics – a simple form that provides the end-user with the details of a metric with a given ID.
- Check files – a simple form that provides the end-user with the details of the file with the given ID.

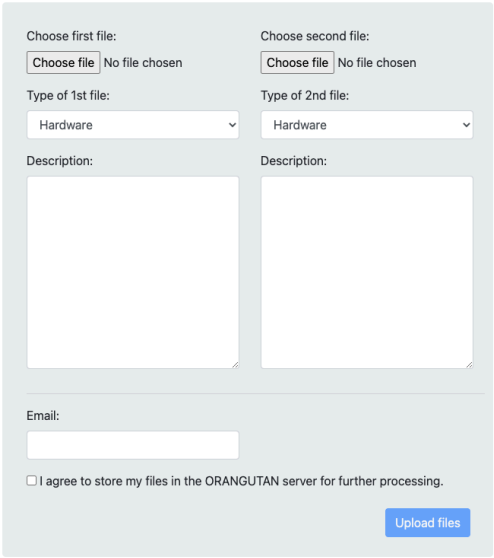


Fig. 5. The layout of the form for uploading sequences.

Once the file is successfully uploaded to the system, its properties can be previewed. Using the *Check file* button on the home page displays a form where the file ID needs to be entered. If the related record is found in the database, the file properties page is displayed. The template of this view is shown in Figure 6. The first part of the template contains a box with explicit file parameters. The second contains a histogram of bytes. The third part shows all bytes of the file.

As shown in Figure 7 metric view contains detailed information of the given metric. Cost and length ratios are marked with a red dot in the common chart where all stored metrics are included. Thus, users can see how their metric compares to all previously calculated metrics. To plot the graph, the Plotly 4.14.3 library is used.

Each metric marked in the graph has a cluster attached. Concerning the description in section II-A, the cluster of each metric is denoted with a color described in sequence: color 1 - cluster LL; color 2 - cluster LH; color 3 - cluster HH; color 4 - cluster HL.

B. Server side

In MVT architecture a core structure is the database. It stores data according to the logic defined by the Model. Database performance has an impact on the speed of operations performed by Django [14]. The engine of the presented application is the database PostgreSQL 12.7 [16]. To allow the application to communicate with PostgreSQL it is also necessary to use the psycopg2 2.8.6 library.

The preparation of appropriate data for machine learning is of particular importance for the experiment methodology because the input data strongly influences the behavior of

File details

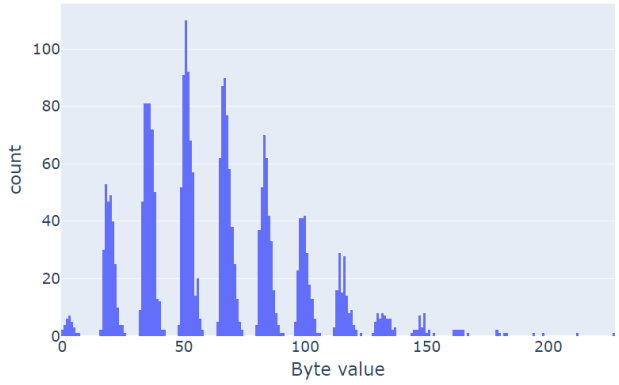
Details of uploaded file: Rayleigh_3_r2.bin with ID: 200

File name: **Rayleigh_3_r2.bin** File type: **SW**

File description: **sztuczne**

File upload as: **uploads/Rayleigh_3_r2.bin** File upload time: **June 2, 2021, 12:55 p.m.** Total number of bytes: **2500**

Histogram of bytes contained in the file



Bytes presentation contained in the file

	1	2	3	4	5	6	7	8	9	10
1	55	37	102	100	34	67	69	129	162	116
2	34	22	66	67	34	116	113	51	52	84
3	33	35	55	35	51	81	117	99	53	38
4	70	67	52	38	104	35	34	71	4	36
5	67	21	34	82	87	20	21	54	83	21

Fig. 6. File details template with bytes histogram and list.

Metric details

Statistical similarity

Details of calculated metric.

Length ratio: **1.6923** Cost ratio: **1762.7122876204503** Cluster: **HL**

Cluster description:
HL. High stretch ratio while maintaining low cost indicates frequent insertion of single gaps, i.e. scattered short matches and mismatches. The result is a premise for inference about the statistical similarity of the sequences.

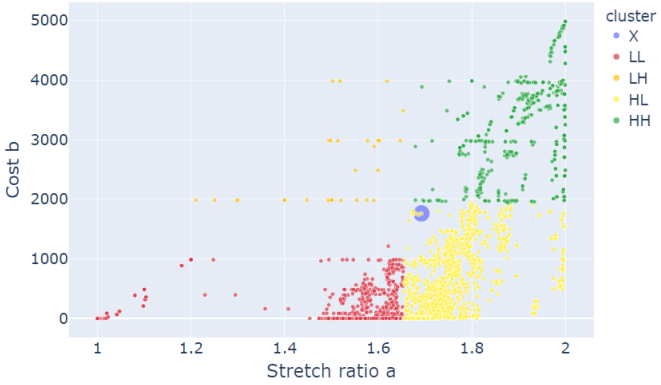


Fig. 7. Metric details template.

individual algorithms and ultimately the obtained results. Several sets of binary data files have been prepared to achieve the most accurate results. The first set contained binary data with different probability distributions. Additionally, 40 files were manually modified by inserting matchless subsequences of known length. Each file was compared to the others, which resulted in 1,600 values on the cost-stretch plane, i.e., metrics indicating the sequence similarity. The second set contained 19 files generated using random number generators implemented in FPGAs and computer simulations of random number generators. The metrics were once again calculated by comparing each file with the rest and an additional 1,521 comparisons were obtained. Acquired metrics were added for a total of 3,191 points on the cost-stretch plane.

In order to implement the cyclic Unsupervised Machine Learning, the data clustering problem needs to be solved first. To find an optimal number of clusters, the KMeans algorithm was used. It performs clusterization based on the sum of squares of the distances of each data point in all clusters to their respective centroids. The resulting curve is shown in Figure 8. Inflection points for $n = 2$ and $n = 4$ were identified. The value of $n = 2$ is the minimum number of clusters as it causes a significant decrease in the Within Cluster Sum of Squares (WCSS) value. On the other hand, $n = 4$ is optimal, as further increasing the number of clusters no longer decreases WCSS. The authors decided to use $n = 4$ clusters.

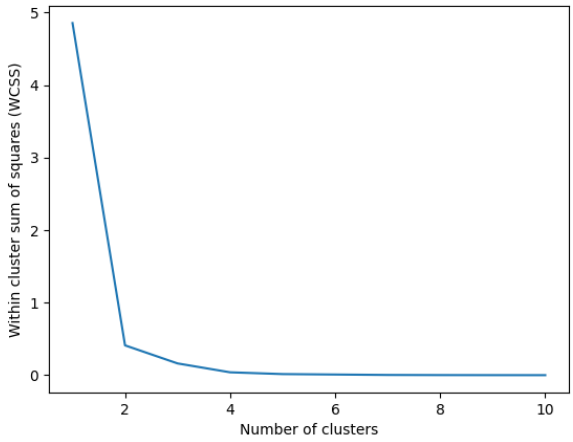


Fig. 8. Sum of squares within the cluster.

While looking for a proper clusterization model, the following algorithms were compared: Affinity Propagation; Agglomerative Clustering; BIRCH; DBSCAN; K-Means; Mini-Batch K-Means; Mean Shift; OPTICS; Spectral Clustering; Mixture of Gaussians [17]. Initially, clustering algorithms used two features, i.e., cost and stretch ratio, however, the data set was divided mainly by the cost, as shown in Fig. 9. According to the best knowledge of the authors, the stretch ratio is also an important parameter that has an analytical interpretation. Therefore, the division of data due to the stretch ratio was forced by second clustering, with only this feature specified. The result is shown in Fig. 10. Then the two split methods were combined. After comparing the clusterization results, the Birch (Fig. 9) and Agglomerative Clustering (Fig. 10) models proved to be the most accurate and consistent with theoretical

assumptions. Taking advances of both algorithms, metrics were clustered and labeled to make Supervised Learning applicable. The resulted final clusterization is presented to the user as previously shown in Fig. 7. Population of designated metrics presented in the Figure 7 contains 3191 points. Each cluster consists of the following number of points: LL - 845 (26.48%), LH - 30 (0.94%), HH - 716 (22.44%), HL - 1600 (50.14%).

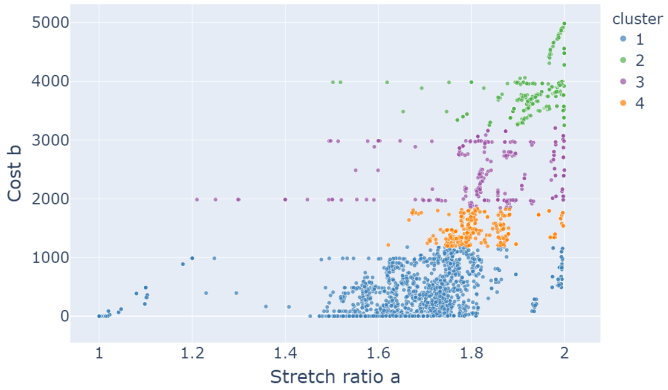


Fig. 9. Clusterization using Birch model for $n = 4$ clusters.

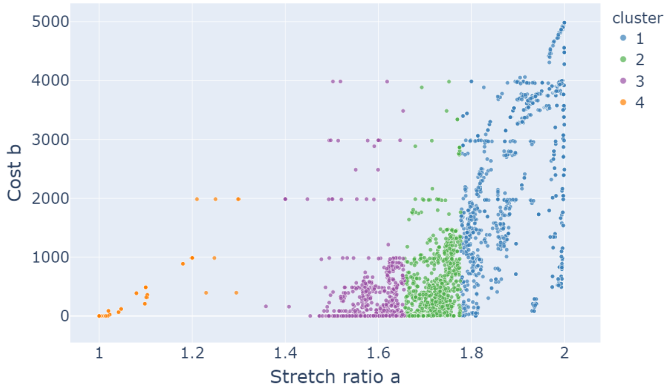


Fig. 10. Clusterization using Agglomerative model for $n = 4$ clusters, only for length ratio.

After data is labeled, the prediction algorithm was trained. For the described problem a Logistic Regression model was chosen.

IV. Conclusions and Further Work

This article presents the proposed Online Platform for Random Number Generator Utility Analysis titled as *ORANGUTAN*. It is a free web tool for convenient testing and easy evaluation of random number generators. The system exploits the original random sequences matching algorithm and as a result, the user receives easily interpretable evaluation metrics including cost and length ratios. Both values are marked on the common graph, placing the designated point in relation to the entire population of metrics. Moreover, the user is able to check properties of the uploaded file, i.e., histogram of bytes and bytes itself. Implementing Semi-Supervised Machine Learning improves interpretation of the results and ready-made evaluation delivered to the user in an accessible form by the frontend. The operation of the pre-trained system has been confirmed experimentally in the evaluation of RO-based hardware generators implemented in FPGAs from leading manufacturers. Further work will mainly concern further data analysis. Simultaneously, the platform will be improved. The development plan assumes mechanisms such as Redis or Celery to be implemented, which will allow for optimal management of memory and tasks performed on the computing server. In the nearest future platform will also be extended to run and evaluate statistical tests such as NIST SP800-22, NIST SP800-90B, DieHarder, and others.

References

[1] D. Li, Z. Lu, X. Zou and Z. Liu, "PUFKEY: A High-Security and High-Throughput Hardware True Random Number Generator for Sensor Networks," Sensors, vol. 10, pp. 26251–26266, 2015.

[2] N. Anandakumar, S. Sanadhya and M. Hashmi, "FPGA-Based True Random Number Generation Using Programmable Delays in Oscillator-Rings," IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 67, no. 3, pp. 570–574, 2020.

[3] F. Bernard, V. Fischer, C. Costea and R. Fouquet, "Implementation of Ring-Oscillators-Based Physical Unclonable Functions with Independent Bits in the Response," International Journal of Reconfigurable Computing, vol. 2012, art. no. 13, 2012.

[4] A. Rukhin, J. Sota, J. Nechvatal et al., "A statistical test suite for random and pseudorandom number generators for cryptographic applications," Special Publication NIST 800-22, National Institute of Standards and Technology, 2010.

[5] M. S. Turan, E. Barker, J. Kelsey et al., "Recommendation for the Entropy Sources Used for Random Bit Generation," Special Publication NIST 800-90B, National Institute of Standards and Technology, 2018.

[6] W. Schindler and W. Killmann, "Evaluation Criteria for True (Physical) Random Number Generators Used in Cryptographic Applications" in Cryptographic Hardware and Embedded SystemsCHES 2002, Lecture Notes in Computer Science, vol. 2523, pp. 431-449, B. Kaliski, C. Koc and C. Paar, Eds. Berlin: Springer, 2003.

[7] P. L'Ecuyer, R. Simard, "TestU01: A C Library for Empirical Testing of Random Number Generators," Association for Computing Machinery, 2007.

[8] R. G. Brown, D. Eddelbuettel and D. Bauer, "Dieharder: A Random Number Test Suite," Accessed on: May 7, 2021. [Online]. Available: https://webhome.phy.duke.edu/~rgb/General/rand_rate.php

[9] J. Nikonowicz, L. Matuszewski and P. Kubczak "Sequence Alignment Algorithm for Statistical Similarity Assessment," arXiv:2106.04349, 2021.

[10] O. Chapelle, B. Scholkopf and A. Zien, "Semi-Supervised Learning," Cambridge, MA, USA: MIT Press, 2010.

[11] J. van Engelen, "A survey on semi-supervised learning," Machine Learning, vol. 109, pp. 373–440, 2020.

[12] Z. Zhou, "A brief introduction to weakly supervised learning," National Science Review, vol. 5, pp. 44-53, 2017.

[13] H. Gore, R. K. Singh, A. Singh, A. P. Singh, M. Shabaz, B. K. Singh, V. Jagota, "Django: Web Development Simple & Fast. Annals of the Romanian Society for Cell Biology," 25(6), 45764585, Retrieved from https://www.annalsofrscb.ro/index.php/journal/article/view/6301, 2021.

[14] Django 3.2 documentation Web page. Accessed: on June 3, 2021. [Online]. Available: https://docs.djangoproject.com/en/3.2/.

[15] Bootstrap 4.6 documentation Web page. Accessed: on May 30, 2021. [Online]. Available: https://getbootstrap.com/docs/4.6/.

[16] Postgres 12.7 documentation Web page. Accessed: on June 3, 2021. [Online]. Available: https://www.postgresql.org/docs/12/.

[17] Documentation of the scikit-learn Python library. Accessed: on June 6, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/clustering.html.