Methods of Searching for Similar Device Fingerprints Using Changes in Unstable Parameters

Marcin Gabryel[1](✉) and Krzysztof Przybyszewski[2,3]

[1] Institute of Computational Intelligence, Czestochowa University of Technology, Al. Armii Krajowej 36, 42-200 Częstochowa, Poland
marcin.gabryel@iisi.pcz.pl

[2] Information Technology Institute, University of Social Sciences, 90-113 Łodz, Poland

[3] Clark University Worcester, Worcester 01610, MA, USA

Abstract. Web-based device fingerprints (also known as browser fingerprints) are designed to identify the user without leaving a trace in the form of cookies. Some institutions believe that this technique violates the privacy of Internet users; however, it allows for an effective fight against fraudsters generating abusive traffic, which brings losses in Internet advertising. Acquiring the parameters that make up a device fingerprint is rather easy as it is done using JavaScript. Most available parameters, however, do not allow for a clear distinction between users or change quite often over time and are therefore considered unstable. This paper presents an algorithm for searching similar web-based device fingerprints, taking into account changing, unstable parameters obtained from the browser and HTTP headers. The presented algorithm is based on the LSH (Locality-Sensitive Hashing) algorithm, which is commonly used to quickly search for similar documents. The effectiveness of the algorithm performance has been checked by using a database of several thousand visits to various websites.

Keywords: Web-based device fingerprint · Browser fingerprint · Locality-Sensitive Hashing

1 Introduction

By using the HTTP mechanism - the so-called cookies, i.e. the ability to store certain information on the client's computer – it is rather easy to identify the user by giving them a unique identifier. Many websites tracking their users use this technique, among other things, to keep track of their current interests and to adjust relevant advertising themes. Nowadays, there is a growing emphasis on privacy, and users are increasingly aware of the possibility of being tracked. This information is provided by the websites themselves, which have been forced to provide information about the use of cookies. There are a number of tools that block user tracking. However, apart from obvious privacy issues, user identification has positive applications. Internet advertising is repeatedly abused in connection with the generation of Internet abusive traffic. This most often manifests itself

L. Rutkowski et al. (Eds.): ICAISC 2020, LNAI 12416, pp. 325–335, 2020.
https://doi.org/10.1007/978-3-030-61534-5_29

in the generation of unnatural clicks, advertisement displays or an automatic entering of personal data into contact forms. Cookies are the easiest method of tracking a user on the Internet and at the same time the easiest one to avoid. Therefore, other methods are used to identify the user online. One method is to generate a so-called web-based device fingerprint (or browser fingerprint), which attempts to identify the user's device and browser on the basis of the unique properties and parameters of the browser. Among the parameters that are obtained are the name and version of the browser and of the operating system, screen information, system font set, canvas fingerprint, HTTP headers, WebGL fingerprint, AudioContext fingerprint, timezone, languages set in the browser, information about installed plug-ins and many others.

The challenge of generating a device fingerprint is to balance fingerprint features with regard to their diversity and stability. The more features are used to create a fingerprint, the more likely it is that all data records will be different. The more traits are used, the more likely it is that individual parameter values will change, thus affecting the stability of a fingerprint. In practice, the values of some features naturally tend to change over time. This is due to browser or operating system updates, installation of new fonts or plugins. Therefore, in most cases, a high level of feature variation reduces the stability. A high level of stability can often only be achieved by including as few features as possible. As far as fingerprint extraction is concerned, it is necessary to find a compromise between diversity and stability.

There are many studies in the literature which describe new methods of obtaining parameters improving the determination of the uniqueness of a given device. At the same time, many organizations (including browser manufacturers) are announcing that the possibility of user identification through fingerprinting mechanisms will be limited. There are also tools available that hinder the process of acquiring parameter values that make up the device fingerprint [8]. One of the first descriptions of the browser fingerprint identification technique appeared in 2011 [3]. The authors proved that by using only a few parameters, such as part of the IP address, font list, time zone, and screen resolution, they were able to discern most users of popular browsers. A major breakthrough was the introduction of canvas fingerprint [4], which based its operation on generating images using the possibilities offered by HTML5. It turns out that graphic processing units generate images in different ways. This allows for a relatively good identification of the user's browser. A similar principle is used to gen-erate AudioContext fingerpint, where it is a property of machine's audio stack itself. This is conceptually similar to canvas fingerprinting: audio signals processed on dif-ferent machines or browsers can vary slightly due to hardware or programming dif-ferences between machines, while the same combination of a machine and browser will still give the same output.

There are a number of studies and compilations providing information concerning the analysis of parameters constituting device fingerprints [5, 6]. One of the most recent studies [1] describes in quite a detailed way the capabilities of the acquired data and the usefulness of creating unique identifiers for browsers. It also contains information about the stability of particular parameters during the tests carried out on working websites. A quite interesting observation from the conducted research is the fact that the stability of collected parameters is maintained for an average of 6 days. However, individual parameters may change earlier.

Another problem is storing such a large and diverse amount of data. For the purpose of a quick search for similar fingerprints, their values are stored in a database in the form of hashes (generated, for example, by the SHA1 algorithm). A pair of fingerprints is considered identical when their hashes are the same. Unfortunately, a change of at least one parameter forming a fingerprint causes the whole hash to change.

The conclusions from the studies described above and the access to several thousand data collected from different types of websites have prompted the author to develop an algorithm that could allow for a comparison of device fingerprints, taking into account parameter changes occurring in particular parameters. The algorithm is based on the popular and fast Locality-Sensitive Hashing (LSH) similar documents search algorithm. The parameters were divided in terms of stability into two groups: one with low or no variability at all and the other with high variability. The algorithm consists of two parts:

- Search for fingerprints potentially similar to the tested fingerprint by using the first group of stable parameters.
- The fingerprints obtained in the search described in the previous point are then searched for target fingerprints using the values of the unstable group parameters. This is done with a much greater degree of probability that the two fingerprint pairs are similar to each other.

The paper presents experimental studies showing the influence of the values of different parameters of the LSH algorithm on the search results, the selection of their optimal values and a comparison with the results obtained from the search conducted using hashing.

The article is divided into the following sections where Sect. 2 describes the LSH algorithm and details of device fingerprinting. Section 3 presents the algorithm for generating and searching for fingerprints. The next section presents the results of the algorithm's performance. The paper ends with conclusions.

2 Algorithms Used in the Research

2.1 Locality-Sensitive Hashing

The main task of the Locality-Sensitive Hashing (LSH) algorithm is to quickly compare documents in terms of their contents. The LSH algorithm consists of three steps:

- transforming the document into a set of characters of length k (the shingling method, also known as k-shingles or k-grams method),
- compressing the shingles set using the "minhashing" method, so that the similarity of the base sets of documents in their compressed versions can still be checked.
- the LSH algorithm, which allows us to find the most similar pairs of documents or all pairs that are above some lower bound in similarity.

Shingling is an effective method of representing a document as a set. To generate the set, we need to select short phrases or sentences from the document, the so-called shingles. This causes documents to have many common elements in their sets even if the sentences appear in documents in a different order.

The next step consists in creating the so-called characteristic matrix, where the columns contain sets of shingles of individual documents, and the consecutive lines correspond to individual shingles. In the matrix cells at the intersection of row i and column j there is value 1 in the case of the i-th shingle in the j-th document.

The idea of hashing is to convert each document to a small signature using hashing function H. If d stands for a document, then $H(d)$ stands for the signature. The H function should be selected so that if the similarity of $sim(d_1, d_2)$ is high, then the probability that $H(d_1) = H(d_2)$ would also be high. If the similarity of $sim(d_1, d_2)$ is small, then the probability that $H(d_1) = H(d_2)$ would be low. In the case when the well-known Jaccard similarity index is used, then MinHash is an appropriate hash function.

In the MinHash algorithm a so-called SIG signature matrix is created with the dimensions $m \times n$, where each of the m documents corresponds to n signatures. The matrix is calculated by performing random and independent n permutations of m rows of the characteristic matrix. The MinHash value for the column of the j-th document is the number of the first row (in the order resulting from the permutations), for which this column has value 1. These calculations are time-consuming, therefore instead of selecting random n row permutations, random n hash functions $h_1, h_2, \ldots, h_n$ are selected. The signature matrix is built taking into account each row in the given order. Let $SIG_{k,j}$ be an element of the signature matrix for the k-th hash function and column j of document d_j. Initially, set $SIG_{k,j}$ to ∞ for all values of k and j. For each row i from the signature matrix, follow these steps:

1. Calculate $h_1(j), h_2(j), \ldots, h_n(j)$.
2. For each column j, check if there is 1 in row i. If yes, then for each $k = 1, 2, \ldots, n$, set $SIG_{k,j} = \min\big(SIG_{k,j}, h_k(j)\big)$.

The idea of the LSH algorithm allows to check similarity of two elements. As a result of its operation, information is returned whether the pair forms a so-called "candidate pair", i.e. whether their similarity is greater than a specified threshold t (similarity threshold). Any pair that hashed to the same bucket is considered as a "candidate pair". Dissimilar pairs that do hash to the same bucket are false positives. On the other hand, those pairs, which despite being similar do not hash to the same bucket under at least one of the hash functions, are false negatives.

A possible approach to the LSH algorithm is to generate hashes for elements several times. For this purpose a suitable signature matrix can be generated:

1. The signature matrix is divided into b bands, and each band is divided into r rows.
2. For each band, pairs of columns that have the same values are checked. If there is such a pair in one band, it becomes a candidate pair and is thrown into the bucket.
3. Parameters b and r should be selected so as to find as many similar pairs as possible, but at the same time as few *false positives* and *false negatives* as possible.

If s is the Jaccard similarity index between a pair of documents, the probability that they are a one-candidate pair equals:

$$p_c = 1 - \left(1 - s^r\right)^b, \tag{1}$$

where s is the Jaccard similarity coefficient defined by the following formula:

$$s = \frac{|d_1 \cap d_2|}{|d_1 \cup d_2|} \tag{2}$$

for two documents d_1 and d_2. A detailed description of particular parts of the LSH algorithm can be found in a number of works including [2].

2.2 Device Fingerprint

Fingerprint is generated on the basis of the parameters provided by a browser and on the basis of hardware capabilities of the device. This operation requires analysis of the JavaScript language of many web browsers, including those running on mobile devices. The algorithm for generating a unique fingerprint identifier given to identify the browser and the device on which the browser is operating consists in collecting as many parameters as possible the browser's API and performing the hash function on them. It is assumed that the generated hash should be unique enough to distinguish between the devices used.

To quantify the level of identifying information in a fingerprint is used the entropy. The higher the entropy is, the more unique and identifiable a fingerprint will be. Let H be the entropy, X – a discrete random variable with possible values $\{x_1, \ldots, x_n\}$ and $P(X)$ - a probability mass function. The entropy follows this formula:

$$H(X) = -\sum\nolimits_i P(x_i) log_b P(x_i). \tag{3}$$

For $b = 2$ it is the Shannon entropy and the result is in bits. One bit of entropy reduces by half the probability of an event occurring.

A device fingerprint is created from many different parameters with a varied number of bits of the entropy. For the tests carried out in this work, the set presented in Table 1 was downloaded from the browser. The data was obtained from 80,000 different devices from which the users accessed different websites. The table does not include parameters with entropy below 0.1. Some parameters, such as *screen_id* and User-agent were broken down into individual elements. For *screen_id* it is *width*, *height*, *available_width* and *available height*. In the case of User-agent the whole sequence was divided into elements starting with prefix *ua*. The parameters were divided into two groups: group 1 – stable parameters, which do not get changed naturally and group 2 – parameters which get changed naturally (unstable).

Table 1. Device fingerprint obtainable features

Feature	No. of bits of entropy	Group	Feature	No. of bits of entropy	Group
device_memory	1.84	1	*browser_plugins_hash*	1.59	2
do_not_track_val_id	0.39	1	*user-agent*	9.80	–
fonts	3.15	1	*br_version*	3.08	2
audio_params_id	0.98	1	*os_version*	3.59	2
webgl_vendor_id	1.84	1	*app_version*	9.79	2
webgl_renderer_id	6.99	1	*platform*	1.74	1
logic_cores	1.64	1	*ua_device_brand_name*	2.52	1
platform_id	1.74	1	*ua_device_model*	4.88	1
timezone	0.43	1	*ua_client_name*	2.17	1
app_version_id	9.79	1	*ua_client_version*	4.03	2
touch_enabled	1.00	1	*ua_client_type*	0.41	1
max_touch_points	1.28	1	*ua_device_type*	1.34	1
screen_id	5.87	–	*ua_device_brand*	2.52	1
width	3.83	2	*ua_device_code*	5.07	1
height	4.50	2	*ua_os_name*	1.21	1
av_width	3.96	2	*ua_os_version*	3.42	1
av_height	5.33	2	*ua_preferred_client_name*	2.56	1
adblock_enabled	0.59	1	*ua_preferred_client_version*	4.31	2
canvas_2d_fingerprint	6.34	1	*ua_preferred_client_type*	0.25	1

3 Proposed Algorithm

Commonly used algorithms used for generating device fingerprint [6, 7] generate a unique fingerprint identifier in the form of a hash, which is an alphanumeric sequence of a fixed length. To this end is used a hash function which generates a short hash as an identifier of a large set of data. Two identical sets of data always generate the same hash value. The prerequisite for selecting a new hash function is to check whether the same hash value exists for different datasets. The identifiers created in this way make it impossible to quickly compare them with each other instead of comparing the data set values. However, commonly used hash functions have one disadvantage, i.e. even a small change of one parameter generates a completely different hash value. In the case of a device fingerprint, many of the values that make up a fingerprint are variable over the length of time during which a given device or browser is used, and they, for instance, include screen sizes or software versions. The purpose of this paper is to try to find such a method of fingerprint encoding so that it would be possible to efficiently determine the similarity of two fingerprints despite minor parameter changes. This in turn will make it possible to identify the browser as the same even if there are changes in the parameters during subsequent visits to the website.

The algorithm is based on the LSH and uses the possibility of adjusting the similarity probability value of two documents. This algorithm needs to have the following operating parameters adjusted – the number of bands b and the resulting number of rows r. The adjustment of these values results from the adopted number of MinHash signatures n encoding the documents and similarity threshold t.

For a given number of signatures n, the choice of b and r depends on value s calculated by using formula (2), following the algorithm:

1. Prepare the signature matrix SIG for a given value of MinHash n.
2. Determine the similarity threshold t.
3. Establish the proportions in the form of weights w_{fp} and w_{fn} between the number of *false positive* and *false negative* samples among the candidate pairs. The following condition needs to be met $w_{fp}, w_{fn} \in (0, 1)$ and $w_{fp} + w_{fn} = 1$.
4. For each possible pair combination b, r find the optimal pair b_t, r_t for which the weighted total of the probability value of *false positive* and *false negative* samples:

$$p_{fp}(t, b, r) = \int_0^t 1 - \left(1 - s^r\right)^b ds \tag{4}$$

and

$$p_{fn}(t, b, r) = \int_t^1 1 - \left(1 - s^r\right)^b ds \tag{5}$$

will be the smallest:

$$b_t, r_t = \operatorname{argmin}_{\substack{b = 1, \ldots, n \\ r = 1, \ldots, n/b}} \left(w_{fp} p_{fp}(t, b, r) + w_{fn} p_{fn}(t, b, r)\right) \tag{6}$$

In the proposed algorithm the fingerprint parameters need to be divided into the stable and unstable ones (see Sect. 2.2). The algorithm is created following the steps presented below:

1. Prepare two sets of parameters: stable ones f_s and unstable ones f_n.
2. Determine the number of signatures for stable and unstable n_s and n_n respectively.
3. Determine the similarity thresholds t_s and t_n.
4. Create two signature matrixes: SIG_s and SIG_n.

In order to find fingerprints similar to fingerprint f_q the following steps need to be carried out:

1. Start the LSH algorithm using the stable parameters to find similar candidate pairs f_{qs}:

$$f_{qs} = \mathrm{LSH}\left(SIG_s\left(f_q\right), SIG_s(f_s), t_s\right) \tag{7}$$

where: $SIG_s\left(f_q\right)$, $SIG_s(f_s)$ – signature matrix values for stable parameters obtained for the parameters of fingerprint f_q and f_s.

2. Having found fingerprints f_{qs} do one more search for similar fingerprint, but this time using unstable parameters:

$$f_{qn} = \mathrm{LSH}\left(SIG_n\left(f_q\right), SIG_n(f_s), t_n\right) \tag{8}$$

where: $SIG_n\left(f_q\right)$, $SIG_n(f_s)$ – signature matrix values for unstable parameters obtained for the parameters of fingerprints f_q and f_s.

3. Return obtained similar fingerprints f_{qn}.

4 Study Results

The study was carried out on the authentic data collected by a specially prepared script run in the browser during the visits made to the website. The script collected data from 26 websites of various types (Internet shops, loan companies, advertising companies, banks and others) for 3 months. During this period of time several hundred thousand data of different browsers were collected. Using cookies it was possible to identify further visits made by the same users. Thanks to this, it was possible to monitor the changes in the parameters of the browsers. For the study, the data were selected from those persons where the changes occurred in 6, 7 or 8 cases during all the visits made. The most frequent changes occurred in the parameters listed in Table 1 as the second group.

According to the algorithm presented in Sect. 3, for the first group of parameters f_s (the stable ones) the search for similar parameters using the LSH algorithm working on the values of the SIGs matrix with the number of signatures $n_s = 128$ and the similarity threshold $t_s = 1$ were applied. For the obtained candidates f_{qs} the LSH algorithm was applied once again in the second group of parameters f_n (the unstable ones). In this step of the algorithm a number of experiments were carried out where the number of signatures n_n of the MinHash algorithm and the similarity threshold t_n were changed. Precision and recall measures were used as a measure of the effectiveness of the conducted studies [14]. Precision is the ratio of the number of correctly classified data to the total number of irrelevant and relevant data classified:

$$precision = \frac{tp}{tp + fp}$$

and recall is the ratio between the number of data that are correctly classified to the total number of positive data:

$$recall = \frac{tp}{tp + fn}$$

where *tp* – true positive, *fp* – false positive, *fn* – false negative and they can be derived from a confusion matrix [14]. The effectiveness of the algorithm performance was analyzed on 100 randomly selected users. The test consisted in determining whether despite the changes occurring in the fingerprint parameters assigning a particular visit to the user that had actually made it (repeat visits) was correct. The obtained precision and recall values are presented in Tables 2 and 3.

For the same data, the method of generating the hash from the acquired parameters was also used. The search for identical devices is therefore a search for identical hash values. In this experiment the obtained values were: precision = 0.53 and recall = 0.13.

In Tables 2 and 3 the results with better precision and recall values than those obtained using only hashes were marked in bold. When analyzing the results one can see that the best results of the comparison of the devices can be obtained for $t_n = 1$, 0.9 or 0.8 and $n_n = 8$.

Table 2. Precision for different probabilities of similarity t_n and MinHash value n_n.

t_n	n_n									
	2	4	8	16	24	32	48	64	96	128
1	0.50	0.52	**0.56**	**0.56**	**0.57**	**0.58**	**0.58**	**0.58**	**0.59**	**0.59**
0.9	0.50	0.52	**0.56**	**0.56**	**0.54**	**0.54**	**0.54**	**0.54**	**0.54**	**0.55**
0.8	0.50	0.52	**0.55**	0.53	0.51	0.52	0.52	0.51	0.51	0.51
0.7	0.50	0.52	0.51	0.49	0.49	0.47	0.48	0.50	0.50	0.50
0.6	0.50	0.46	0.47	0.48	0.47	0.47	0.47	0.46	0.46	0.46
0.5	0.47	0.46	0.46	0.46	0.45	0.46	0.46	0.45	0.46	0.45

Table 3. Recall for different values of probability t_n and value of MinHash n_n.

t_n	n_n									
	2	4	8	16	24	32	48	64	96	128
1	**0.19**	**0.14**	**0.14**	0.13	0.13	0.13	0.13	0.13	0.13	0.13
0.9	**0.19**	**0.14**	**0.14**	0.13	0.13	0.13	0.13	0.13	0.13	0.13
0.8	**0.19**	**0.14**	**0.14**	**0.14**	**0.14**	0.13	0.13	0.13	0.13	0.13
0.7	**0.19**	**0.14**	**0.14**	**0.14**	**0.14**	**0.14**	**0.14**	**0.14**	**0.14**	**0.14**
0.6	**0.19**	**0.24**	**0.17**	**0.14**	**0.15**	**0.14**	**0.15**	**0.14**	**0.15**	**0.17**
0.5	**0.27**	**0.24**	**0.25**	**0.20**	**0.22**	**0.16**	**0.19**	**0.19**	**0.19**	**0.20**

5 Conclusion

The paper presents a quick method of comparing device fingerprint using the LSH algorithm. It requires creating hashes using the MinHash method and saving them to the database. Proper selection of parameters of the LSH algorithm requires creating several columns in a table. Indexing these columns will then allow for an easy comparison of the values of subsequent fingerprints.

The algorithm can be extended by using neural networks [11, 15, 18] with an appropriate network structure [10, 13], the big data algorithms [9], fuzzy methods [12, 16] and other [17]. There are also plans for using such parameters as: IP number, Internet Service Provider (ISP), geolocation and additional parameters that can be obtained from the browser of a given device. The problem of similarity of the same mobile phone models whose browsers return exactly the same parameters remains yet to be solved.

References

1. Kobusińska, A., Pawluczuk, K., Brzeziński, J.: Big Data fingerprinting information analytics for sustainability. Future Generation Comput. Syst. **86,** 1321–1337 (2018)
2. Leskovec, J., Rajaraman, A., Ullman, J.D.: Mining of Massive Datasets. Cambridge University Press, Cambridge (2014)
3. Boda, K., Földes, Á.M., Gulyás, G.G., Imre, S.: User Tracking on the Web via Cross-Browser Fingerprinting. In: Laud, P. (ed.) NordSec 2011. LNCS, vol. 7161, pp. 31–46. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29615-4_4
4. Mowery, K., Shacham, H.: Pixel perfect: Fingerprinting canvas in HTML5. In: Proceedings of W2SP, pp. 1–12 (2012)
5. Acar, G., Eubank, C., Englehardt, S., Juarez, M., Narayanan, A., Diaz, C.: The web never forgets: persistent tracking mechanisms in the wild. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, pp. 674–689, November 2014
6. Laperdrix, P., Rudametkin, W., Baudry, B.: Beauty and the beast: Diverting modern web browsers to build unique browser fingerprints. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 878–894. IEEE, May 2016
7. https://github.com/Valve/fingerprintjs2. Accessed 06 Feb 2020
8. Englehardt, S., Narayanan, A.: Online tracking: a 1-million-site measurement and analysis. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 1388–1401, October 2016
9. Koren, O., Hallin, C.A., Perel, N., Bendet, D.: Decision-making enhancement in a big data environment: application of the K-means algorithm to mixed data. J. Artif. Intell. Soft Comput. Res. **9**(4), 293–302 (2019)
10. Shewalkar, A., Nyavanandi, D., Ludwig, S.A.: Performance Evaluation of Deep Neural Networks Applied to Speech Recognition: RNN, LSTM and GRU. J. Artif. Intell. Soft Comput. Res. **9**(4), 235–245 (2019)
11. Ludwig, S.A.: Applying a neural network ensemble to intrusion detection. J. Artif. Intell. Soft Comput. Res. **9**(3), 177–188 (2019)
12. D'Aniello, G., Gaeta, M., Loia, F., Reformat, M., Toti, D.: An environment for collective perception based on fuzzy and semantic approaches. J. Artif. Intell. Soft Comput. Res. **8**(3), 191–210 (2018)
13. Liu, J.B., Zhao, J., Wang, S., Javaid, M., Cao, J.: On the topological properties of the certain neural networks. J. Artif. Intell. Soft Comput. Res. **8**(4), 257–268 (2018)
14. Leskovec, J., Rajaraman, A., Ullmanm, J.D.: Mining of Massive Datasets. Cambridge University Press, Cambridge (2014)
15. Bilski, J., Wilamowski, Bogdan M.: Parallel levenberg-marquardt algorithm without error backpropagation. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2017. LNCS (LNAI), vol. 10245, pp. 25–39. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59063-9_3

16. Korytkowski, M., Senkerik, R., Scherer, M.M., Angryk, R.A., Kordos, M., Siwocha, A.: Efficient image retrieval by fuzzy rules from boosting and metaheuristic. J. Artif. Intell. Soft Comput. Res. **10**(1), 57–69 (2020)
17. Wróbel, M., Starczewski, Janusz T., Napoli, C.: Handwriting recognition with extraction of letter fragments. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2017. LNCS (LNAI), vol. 10246, pp. 183–192. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59060-8_18
18. Gabryel, M., Grzanek, K., Hayashi, Y.: Browser fingerprint coding methods increasing the effectiveness of user identification in the web traffic. J. Artif. Intell. Soft Comput. Res. **10**(4), 243–253 (2020)