Using Bloom filters in data leak protection applications

Sergey Butakov

Information Security and Assurance Department,
Concordia University College of Alberta, Edmonton, AB, Canada
sergey.butakov@concordia.ab.ca

Abstract. Data leak prevention systems become a must-have component of enterprise information security. To minimize the communication delay, these systems require fast mechanisms for massive document comparison. Bloom filters have been proven to be a fast tool for membership checkup with some allowed level of false positive errors. Taking into account specific needs of fast text comparison this paper proposes modifications to the Matrix Bloom filters. Approach proposed in this paper allows to improve density in Matrix Bloom filters with the help of special index to track documents uploaded into the system. Density is improved by combining a few documents in one line of the matrix to reduce the filter size and to address the problem of document removal. The experiment provided in the paper outlines advantages and applicability of the proposed approach.

Keywords: Data Leak Protection, Bloom Filters, Text Search

1 Introduction

Data leak protection (DLP) systems become a must-have part of enterprise information security. One of the main functions for these systems is content filtering. Content filtering in DLP is different from the one in antivirus or spam protection applications. In DLP it relies on the internal sources for comparison rather than on signature database maintained by let's say antivirus supplier. Simple content filtering in DLP could be based on keyword screening. More advanced mechanisms allow comparing text in question with a set of documents that are listed for internal purposes only. It is assumed that comparison must be done on the level of paragraphs or sentences and thus lead us to the task of similar text detection. This task of fast detection of near-duplicate documents is not new. It has been studied for years, starting with applications in simple text search in the 1970s [1]. Additional momentum has been added later by massive projects in genome sets comparison and various research fields in internet study. The latter includes near –duplicate web page detection [2], filtering of web addresses [3], internet plagiarism detection [4], and many others. Although there are many methods to for fast text comparison many of them are not suitable for large scale tasks. One of the approaches called Bloom Filters offer speed of comparison while generating some false positive results as a trade-in for the speed.

Bloom Filters (BFs) were introduced in 1970 as a tool to reduce the space requirements for the task of fast membership checkup [5]. As Bloom mentioned in his paper, this method allows significant reduction in the memory requirements at the price of small probability of false positive results [5]. He also indicated that some applications are tolerable to false positives as they require two-stage comparison: the first step is to quickly identify if an object is a member of a predefined set and the second step is to verify the membership. Text comparison can be considered as one of such tasks when on the first phase the system must reduce the number of documents to be compared from hundreds of thousands to thousands. This step reduces the workload for the second phase of comparison where the system needs to find and mark similar segments in the documents.

1.1 Motivating applications

Insider misuse of information is considered to be the top reason for data leaks [6]. Regardless of the fact that such misuses can be either intentional or unintentional, it is always good to have additional defense mechanisms embedded into the information flow to limit the impact of penetration [7]. Commercial Data Leak Prevention (DLP) systems are developed by major players on information security market such as Symantec or McAfee. One of the tasks for such systems is to make sure that documents belonging to the restricted group do not leave the network perimeter of an organization without explicit permission from an authorized person [8]. DLP systems can work on different layers. On application layer they can analyze keywords, tags or the entire message [9]. The decision to allow the document to be transmitted outside should be made quickly and should be prone to false negative errors. False positive could be acceptable up to the certain level if there are embedded controls in the system for manual verification and approval. An additional desired feature for DLP systems is the ability to detect obfuscated texts where some parts of the text are not confidential but other parts are taken from the restricted documents. BFs could play a preselector role that would facilitate fast search by picking from the large archive of protected documents some limited number of candidate sources that could be similar to the document in question. A detailed comparison can be done using other methods that could be slower but more granular.

1.2 Contribution

BFs have been studied in details in the last four decades. In this paper we reviewed the problems related to their practical implementation. In its original form, the BF can identify that the object is similar to one or more objects in a predefined set of objects but it cannot identify which one it is similar to. In other words BFs do not contain localization information. Matrix Bloom Filter (MBF) [10, 11] allows such identification but, as shown in the next section, may impose significant requirements on memory to maintain the set of documents. Dynamic BFs [12] allow better memory utilization and deletion of elements, but still lack localization property. Approach proposed in this paper allows density improvement for an MBF by constructing spe-

cial index to track documents uploaded to MBF. In addition to index, the document removal algorithm with localized workload has been proposed.

Experiment in the last section shows that compacting documents in the MBF can significantly reduce memory consumption by the filter at the cost of maintaining smaller index of the documents that have been uploaded into the filter.

2 PREVIOUS WORKS

The first part of this section discusses original BF and some major modifications that occurred to improve its performance and usability; the second part highlights the areas where BFs can be improved in applications related to near-similar document detection.

2.1 Bloom Filters with Variations

BF was introduced as memory-efficient method for the task of fast membership checkup [5]. The main idea of BF is to use randomized hash functions to translate an object from set S= $\{x_0, x_1, \ldots, x_{(n-1)}\}$ into a binary vector of fixed size m. BF employs k independent hash functions $h_0, h_1, \ldots, h_{k-1}$ to map each element to a random number over the range $\{0, \ldots, m-1\}$. To insert element x into BF, the corresponding bits of BF – $\{h_i(x), 0 \le i \le k-1\}$ – have to be set to 1. To check if element x' is already in BF the corresponding values of $\{h_i(x'), 0 \le i \le k-1\}$ must be calculated and compared with the values in the filter. If all corresponding bits are set in 1 than x' already exists in the filter. Due to randomized nature of h_i BF may produce a false positive [5]. The probability of false positive can be estimated as follows [13]:

$$p' = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m} \quad (1)$$

As it can be seen from (1) the false positive probability can be reduced by increasing the filter size m. The optimal size of BF can be defined using n - number of elements in S and acceptable risk of false positive - p'. F has distinctive features:

- BF has constant insert and checkup time due to the fixed size of the filter – constant m. This feature makes it an attractive tool to check membership in large collections because checkup time does not grow along with the collection size.
- Removal of any element from BF requires the entire filter to be recreated. The fact that a single bit in the filter can be set to *1* by many elements leads to the necessity to recreate the entire filter if one element has to be removed.
- If the number of elements in set $\{S\}$ grows above n then the new value for m has to be calculated and filter must be recreated.

The last four decades have brought many variations to BFs, aiming to overcome problems of member elements removal, filter fixed size limitations, and some others. Counter BFs [14] use bit counters instead of a single bit value in the filter. This approach allows to store the number of times the particular index has been initialized.

The element addition/removal to/from the filter can be done by increasing/decreasing corresponding counter. Although such an approach keeps the membership checkup operation simple and computationally effective, it increases memory requirements. For example, one byte counter that can store up to 255 initializations of the particular index would increase the BF memory consumption by the factor of 8.

Dynamic or matrix Boom filters (MBF) [12] handle the growing size of $\{S\}$ by adding additional zero-initialized vectors of size m to the filter when number of elements in S reaches n. Each new vector is considered a brand new zero-initialized BF. Such an approach allows indefinite growth of S at the cost of additional checkups because the membership has to be checked in each row in the matrix. The number of checkups required can be considered as linear to the size of S and it is of $\theta(|\{S\}|/n)$ order. Removal of any text would cause the corresponding row to be removed from the matrix.

For the applied problems of document comparison, the speed of comparison can be considered as a priority metric, while the speed of update operations such as addition and deletion of elements can be less of the priority. This metric selection is based on the assumption that a set of documents for the comparison is relatively steady if compared to the information flow that has to go through the filter.

2.2 Related Problems

Near to duplicate document detection usually works on the level of a few consecutive words or a string sequence of the same length without selecting word from the test. Such granularity allows to locate meaningful similarities in the texts while staying prone to minor text alternations. The sequence of words or characters can be called a shingle or a grammar [15, 16]. A full set of these sequences defines text $T = \{t_0, t_1, \ldots, t_{z-1}\}$. If sequencing is done on the word level, then z is close to the number of words in the document. If sequencing is done on the character level it could lead to excessive representations. Many algorithms with proven performance and robustness have been developed to address this issue. For example, Winnowing algorithm provides $d=2/(w+1)$ density of the selection where w is number of characters in the sequence (w-gram) [16].

If we are to compare two documents, T_i and T_j of length z_i and z_j respectively, and the comparison has to be done using BF, then m must satisfy the following inequality: $m \geq max\{z_i, z_j\}$. Obviously, if BF is to be used to compare one document T' with many documents $\{T\}$ then BF has to be large enough to fit all of these many documents. Straightforward sequencing of all the documents followed by placing shingles into the BF will not work because classical BF is not local. In other words, if two documents contain exactly the same shingle (sequences of words or characters) and both documents were placed in BF, the same shingle flips to *1* same bits and therefore there will be no way to identify the source document. It is obvious that similar problem appears when there is a need to remove a document from the comparison set $\{T\}$. As Geravand & Ahmadi suggested, the localization problem can be addressed by matrix BF.

They proposed to use a set of BFs where each BF represents shingles from one document [11]. The document that is scheduled for the checkup generates its own BF which will be compared with every row in matrix BF on the next step. Although a set of independent BFs does not look like a typical BF, it has one important property: the number of *XOR* operations to check the membership in the entire matrix BF is linear to the matrix size. This important performance property is achieved by using the same fixed *m* for all rows of matrix BF.

This approach has the disadvantage of inefficient memory consumption by each row in the matrix because of two factors: the requirement to have one document per row and the requirement to have constant *m*. The last one provides the possibility to calculate BF for *T'* only once and therefore dramatically improve the comparison speed. The first requirement can be relaxed with the additional indexing added to the matrix. The following section proposes such an index and outlines related algorithms.

3 PSEUDO-MATRIX BLOOM FILTER

Placing all the shingles for the entire document collection in one large BF is not an option in DLP. These systems require not only the confirmation that a particular shingle from the document in question belongs to the collection but also require linking the same shingles to the potential source document. The proposed data structure resembles the one proposed by Geravand & Ahmadi for the task of plagiarism detection. Pseudo-Matrix BF (PMBF) will consist of an unlimited number of BFs of the same size [11]. The major alternation is to use one BF to store more than one document where it is possible. From the equation (1) we can find *m* that would allow to limit *p* for a maximum of *n* elements [13]:

$$m \geq -\frac{n\ ln(p)}{(ln(2))^2} \qquad (2)$$

Keeping in mind that *m* is fixed for the entire PMBF to achieve the fast comparison speed, we can state that if there is a restriction to place one and only one document in single BF we can meet the following cases:

- Number of shingles in a document exceeds *n*, thus the entire document cannot be placed in a single row;
- Number of shingles in a document is much less than *n*, thus the BF is underpopulated which makes it memory inefficient.

Suboptimal memory allocation would require *p* to be slightly less than *n*. To provide the such memory allocation by PMBF, two additional indexes have to be maintained: one to keep track of the remaining capacity of each row and another one to store indexes of documents placed in each row of PMBF. Figure 1 provides an example of a PMBF. The example assumes that each row in BLOOM FILTERS matrix is populated with two documents *(T)*. The example can be extended to any number of documents in a single row. INDEX part of the structure links filter rows with uploaded texts $\{T_i\}$. For each row the corresponding INDEX list identifies the documents

that are stored in this row. For example, Figure 1 assumes that each document is stored in two rows. As the last part of the paper shows in real applications there will be no such even distribution and many documents could be stored in one row. If *BF'* compiled from *T'* has a certain level of similarity with BF_i we can say that *T'* is probably similar to one of the documents *(T)* located in BF_i. The detailed comparison must be done to confirm this. It can be done with any appropriate method such as Levenstein's distance; this stage of comparison is out of scope of this paper.

The proposed data structure allows relatively easy document removal with no increase in memory consumption. If T_i to be removed, then only the row(s) that contain contains T_i has to be recreated. For example, in Figure 1 if document T_{1000} are to be removed from the collection then only BF_{h-1} and BF_h have to be recreated making it relatively minor work from the computational perspective. Thus the additional work is not very computationally expensive if comparing with matrix BF presented in [11] but there is higher shingle density in the matrix. The density of each individual filter in the matrix can be evaluated using the index. Under-populated filters can be filled up with additional text segments.

BLOOM FILTERS		**INDEX**
BF_0	----	

BF_{h-2}: T_{998}, T_{999}	----	I_{998}; I_{999};
BF_{h-1}: T_{999}, T_{1000}	----	I_{999}; I_{1000};
BF_h: T_{1000}, T_{1001}	----	I_{1000}; I_{1001};
BF_{h+1}: T_{1001}, T_{1002}	----	I_{1001}; I_{1002};

BF_{d-1}	----	

Fig. 1. Example of Pseudo-Matrix Bloom Filter with two documents in a row

The proposed architecture has the following important features:

- Compared to an MBF proposed earlier, it increases density for each BF_i, therefore increasing its memory efficiency.
- Increased density leads to better computational efficiency of the data structure. Although the comparison computations are still of $\theta(n*d)$ order but in this structure d is the number of rows required to allocate the document corpus instead of total number of documents. As the experiment below shows it could be few times less than the total number of documents.
- It keeps number of binary checkups linear to the size of the collection *{T}*.
- It allows relatively easy document removal.
- Matrix size is pseudo linear to the total number of shingles in the collection.

Example calculations of the size and density provided below show the memory efficiency of the proposed approach. Second part of the experiment shows applicability and limitations of the proposed approach.

4 EXPERIMENT

Two experiments have been conducted to test the proposed approach of PMBF utilization. First experiment has been aimed to evaluate the compression ratio of the PMBF and second experiment has been conducted to study applicability of PMBF on real data.

4.1 Evaluation of memory consumption by PMBF

The goal of the first experiment was to show on the real data that PMBF has distinctive size advantage over matrix BF. In this case the experimental calculations have been performed on the simulated plagiarism corpus used for the PAN'2009 plagiarism detection competition [17]. The corpus contained 14,428 source documents simulating plagiarism. The average number of words in a document is about 38,000; 95% of all documents have less than 123,000 words. The calculations provided in the Table 1 were done with the following assumptions:

- Text shingling is done by sliding window on the word basis. Keeping in mind that shingles (consecutive phrases of *y* words) are relatively short, we can safely assume number of shingles to be equals to the number of words in the text. The number of words in a shingle may affect the granularity of the search in PMBF but this question is out of scope of this paper.
- The order in which the documents are placed in the filter does not affect its memory capacity. In practice, this will not be absolutely true because if random documents are constantly being added and deleted to/from PMBF, it could lead to a fragmented index thus decreasing the performance of the index search. The issue can be addressed by setting up some threshold to protect under-populated rows from accepting short sequences of shingles. This option will be explored in the future research.

As it can be seen from Table 1, for PAN'09 corpus PMBF has compressed the information in 1:3.1 ratio comparing to MBF because the straightforward implementation of one text per row approach would require 14,428 rows in matrix instead of 4489. Moreover, in MBF about 5% of the documents would not fit a single row and therefore would be out of search scope. The option to increase n up to the size of the largest document in the corpus (~428,000 words) would increase an MBF size by the factor of 4.

The compression ratio of 1:3.2 also indicates that 95% of the documents will be collated in one or two rows of the matrix. The latter case covers those documents that start in one row and end in the next one; therefore, in 95% of cases only two rows will

have to be recreated if such a document is scheduled for removal from the filter. On the related issue of space reuse, we can suggest that algorithms similar to garbage collection can be implemented to take care of the released space after a document has been removed from PMBF. The garbage collection can be implemented in the system's off-peak time when search results can be delayed.

Table 1. Filter Size and Operations for the Corpus.

Value	Explanation
p=0.05	5% false positive probability
n=123,000	Filter of this size will accommodate 95% of texts. If text has more than 123,000 words it will be placed in more than one filter. If text has less than 123,000 words empty space will be occupied by another text.
m=766,933 bit.	See equation (2). 766,933 bite ~= 94 kilobytes
d=4489	Total number of rows in PMBF = Total number of words in corpus / Number of words to fit in one row
M ~=411 Megabytes	Total size of PMBF to accommodate PMBF for the entire corpus of 14,428 documents.

Obviously, improvement in space allocation comes with a cost of more comparisons to be done to confirm the similarity of the document, if an MBF similarity of *BF'* and BF_i means that respective texts *T'* and T_i must be compared more closely. In the case of PMBF, the similarity of BF' and BF_i means that *T'* must be compared with more than one document. According to the example above, in 95% of cases on average *T'* will have to be compared with a minimum of 4 documents and a maximum of 8 documents because 95% of the documents are located in one or two rows. Therefore, on one hand increasing *n* helps to improve the compression, but it also increases the workload for the system component that performs detailed comparison.

4.2 Applicability of PMBF for DLP tasks

Goal of the second experiment was to test the applicability of PMBF for text similarities detection. As it was mentioned earlier, DLP systems have to be prone to false negatives and keep false positives in reasonable range. In our case PMBF is used on the first stage of comparison which is to indicate the potential similarity among documents thus reducing the range of the detailed comparison from thousands of potential matches to more manageable number which can be processed on the second page by more precise but slower methods.

Two data sets were used for the evaluation. One data set included 500 documents with text inserts from other 50 source documents. Each randomly placed insert had been taken from only one of 50 sources. The length of the inserts varied from 5% to 50% of the total document length. Each insert was placed as one piece in the random

place of the document. All the documents consisted of about 4000 words. Due to the fact that the documents in this corpus were about the same length the compression was not implemented in this part of the experiment and therefore each document had been placed in the separate row of PMBF. Each of 50 source documents has been compared with all 500 documents from this corpus. The level of similarity was evaluated as number of ones in the same cells of the one row [11]. It was naturally expected to see some similarity among totally different documents as BFs are tailored to have some level of false positives. The question was to see if the actual inserts produce higher level of similarity which is distinct from the background similarity levels. The experiment indicated that for all 50 sources it was true. For each source document the documents that had some parts from that source produced distinctive pikes on the similarity level graph. An example of such graph for one of the sources is presented on Figure 2. It shows 10 distinct similarity levels for 10 sources that included from 5% (leftmost pike) to 50% (rightmost pike) of the text from that particular source. Based on this part of the experiment we can state that PMBF are suitable for preliminary comparison of text documents in the cases where large portions – 5% or above – were copied without variations from one of confidential documents.

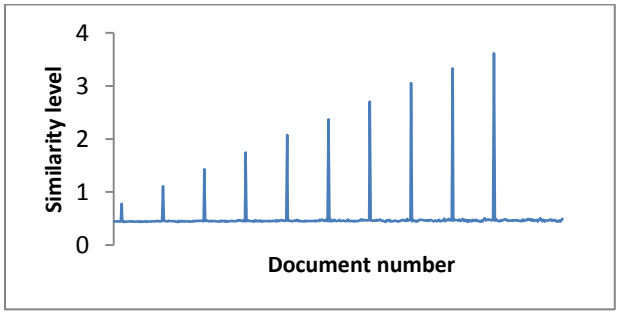

Fig. 2. Example of source document comparison for the first corpus of 500 documents.

Second phase of experiment was conducted to evaluate level of false positives and false negatives on the larger corpus with obfuscated text. Training corpus from PAN'09 plagiarism detection competition [17] was used for PMBF performance evaluation. Although classic BF is prone to false negatives this may not be the case for PMBF. When maximum size of a document – *n* in equations (1) and (2) - is being used to allocate bit string, it represents maximum number of shingles in the document. But in fact the proper estimation of the number of all potential elements in one row should take into account all possible shingles. For example, if triplet of words is being used as a shingle than proper estimation of n would be all possible triplets from the dictionary: dictionary of 10^4 words would generate about 10^{12} combinations. Therefore estimating *n* as maximum number of shingles in a document increases false positives. Such increase eventually will lead to false negatives as less strong comparison criteria will be used to reduce number of false positives.

PAN'09 corpus consists of 14,428 documents [17]. Number of pair-wise comparisons for all the documents in the corpus would be about $(14*10^3*14*10^3)/2 \sim 10^8$.

PMBF will be used on the first phase of DLP checkup process to reduce this number to at least thousands or less. The comparison process was done using two approaches to populate PMBF. In first approach one document was placed in one row only but each row may contain many documents. In the second approach longer documents were distributed among many lines. As it can be seen from Table 2 second approach leaded to much less false positives but as a trade in for the better compression number of false negatives doubled. Additionally in the second approach each row in the PMBF were populated only up to 50% of its potential capacity to reduce false positives. Both approaches produced some false negative results that are not desirable for DLP systems. Detailed study of false negatives indicated that all of them were caused by the documents that contained less than 1% of the text from the source documents. Moreover 26 documents out of 117 false negatives text inserts from the source documents were highly obfuscated. Since the comparison was done on the word level without any additional tools to tackle text obfuscation we can state that these false positives were expected. Two methods produced different amount of false positives. In the second case when density of BF was reduced by 50% the number of false positives decreased significantly. This feature of PMBF gives DLP developers additional choice – if they are ready to use twice as much memory for PMBF then the second stage of comparison will be much less loaded because of much lower level of false positives.

These two experiments indicated that PMBF can be used in DLP if it is acceptable that only larger portions of the restricted documents will be filtered out. This leaves the attackers with the potential to split the document into the smaller chunks to avoid filtering. This would be very suitable when DLP is intended to protect users from errors such as typos in the email address or other mistakes where users do not have intentions to violate the document distribution rules.

Table 2. False positive and false negative results of the experiment for two approaches to populate PMBF

	Document in one line	Distributed documents
False positive results	301977 (~1%)	9322 (~0.01%)
False negative results	51 (~5%)	117 (~11%)

5 CONCLUSION

The paper proposes to use improved data structure based on Bloom filter to address the issue of fast document comparison. The proposed data structure and algorithms allow better memory allocation in Bloom filters aimed on document comparison. Additional index allows BF to locate the potentially similar documents even if few documents have been placed in a single row of the filter. This index also allows computationally effective document removal operations.

One limitation of the proposed approach is related to the fact that using PMBF makes sense only if the entire filter could be allocated in the computer RAM where fast bitwise comparison is possible. Placing parts of the filter on the disk will fade its speed advantage. Based on the experiment above we can state that even minimal server configuration with few Gigabytes of RAM can handle hundreds of thousands of documents which seems to be suitable for DLP systems for a medium enterprise. As first experiment shows PMBF provides noticeable size advantage over matrix BF. Second experiment indicated that PMBFs are applicable for filtering in DLP if document in question includes larger portion (above 5%) of the restricted document. This limitation may not be a problem depending on the purposes of the specific DLP.

ACKNOWLEDGEMENTS

Author would like to acknowledge productive discussions on Bloom Filters with Dr. A. Tskhay, and Mr. V. Shcherbinin as well as help with experiments from Mr. L. Shi and Mr. V. Dyagilev.

REFERENCES

1. Knuth, D., Morris, Jr., J., and Pratt, V. Fast Pattern Matching in Strings. SIAM Journal on Computing 1977 6:2, 323-350 (1977)
2. Brin S., Davis J., and García-Molina H. Copy detection mechanisms for digital documents. SIGMOD Rec. 24, 2 (May 1995), 398-409. DOI=10.1145/568271.223855
3. Cormack G. V. (2008). Email Spam Filtering: A Systematic Review. Found. Trends Inf. Retr. 1, 4 (April 2008), 335-455. DOI=10.1561/1500000006 (1995)
4. Butakov S. and Scherbinin V. The toolbox for local and global plagiarism detection. Comput. Educ. 52, 4 (May 2009), 781-788. DOI=10.1016/j.compedu.2008.12.001 http://dx.doi.org/10.1016/j.compedu.2008.12.001.(2009).
5. Bloom B. H. Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13, 7 (July 1970), 422-426. DOI=10.1145/362686.362692. (1970).
6. Liu, S.; Kuhn, R. Data Loss Prevention. IT Professional , vol.12, no.2, pp.10-13, March-April 2010 doi: 10.1109/MITP.2010.52. (2010).
7. Blackwell C. A security architecture to protect against the insider threat from damage, fraud and theft. In Proceedings of the 5th Annual Workshop on Cyber Security and Information Intelligence Research (CSIIRW '09), Article 45 , 4 pages. DOI=10.1145/1558607.1558659 http://doi.acm.org/10.1145/1558607.1558659. (2009).
8. Lawton G. New Technology Prevents Data Leakage. Computer 41, 9 (September 2008), 14-17. DOI=10.1109/MC.2008.394 http://dx.doi.org/10.1109/MC.2008.394. (2008).
9. Potter B. Document Protection: Document protection. Netw. Secur. 2008, 9 (September 2008), 13-14. DOI=10.1016/S1353-4858(08)70108-9 http://dx.doi.org/10.1016/S1353-4858(08)70108-9. (2008).
10. Wang J.; Xiao M.; Dai Y. "MBF: a Real Matrix Bloom Filter Representation Method on Dynamic Set," Network and Parallel Computing Workshops, 2007. NPC Workshops. IFIP International Conference on , vol., no., pp.733-736, 18-21 Sept. 2007 doi: 10.1109/NPC.2007.107 (2007).

11. Geravand, S.; Ahmadi, M. A Novel Adjustable Matrix Bloom Filter-Based Copy Detection System for Digital Libraries, Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on , vol., no., pp.518-525, Aug. 31 2011-Sept. 2 2011 doi: 10.1109/CIT.2011.61 (2011)
12. Guo, D.; Wu, J.; Chen, H.; Luo, X.; Theory and Network Applications of Dynamic Bloom Filters, INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings , pp.1-12, April 2006 doi: 10.1109/INFOCOM.2006.325 (2006)
13. Broder A.Z. and Mitzenmacher M. Network Applications of Bloom Filters: A Survey. Internet Mathematics, Vol 1, No 4, 2005. P485-509 (2005).
14. Fan L., Cao P., Almeida J., Broder A. 2000. Summary cache: a scalable widearea web cache sharing protocol. IEEE/ACM Trans. on Networking, vol.8, no.3, pp.281-293, (2000).
15. Karp R. M. and Rabin M. O. Pattern-matching algorithms. IBM Journal of Research and Development, 31(2):249–260, 1987. (1987).
16. Schleimer S., Wilkerson D., and Aiken A. Winnowing: Local Algorithms for Document Fingerprinting. Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 76-85, June 2003. (2003).
17. Potthast M., Eiselt A., Stein B., Barrón-Cedeño A., and Rosso P. PAN Plagiarism Corpus PAN-PC-09. http://www.uni-weimar.de/medien/webis/research/corpora, (2009)