A Novel Adjustable Matrix Bloom Filter-based Copy Detection System for Digital Libraries

Shahabeddin Geravand
Department of Computer Engineering
Islamic Azad University of Arak
Arak, Iran
shgeravand@gmail.com

Mahmood Ahmadi
Department of Computer Engineering
University of Razi
Kermanshah, Iran
m.ahmadi@razi.ac.ir

Abstract—With the increasing volume of on-line literatures on the Internet and the simplicity of finding and downloading data, dishonest use of the findings of others, known as plagiarism, is getting worse and worse. Therefore, there is a need to be a copy detection system to address this problem in an efficient way. Most current systems only focus on one goal, estimating similarity with highest accuracy, i.e. 100%. While, in some real applications, it can be useful to take into account other factors such as query speed, memory usage and security of content at the cost of reducing accuracy by a few percentages. In this paper, we propose an innovative adjustable copy-paste detection system which provides an adjustable property on mentioned factors according to the application requirements. The main core of our design is a new extension of Bloom filters, called Matrix Bloom Filter (MBF), which provides the adjustability of the system. A matrix Bloom filter is defined as a bit matrix in which each entry can only be set or reset. It is utilized to efficiently maintain all documents of libraries. Based on our knowledge, this is the first work using the idea behind Bloom filters to solve copy-paste detection problem while ensuring the privacy of document content and also the first work aiming to provide this adjustable property. The experimental results show that our proposed approach provides three main improvements, including enhancing the speed of querying operation up to 2.7 times, diminishing the memory required and providing the security of content besides allowing an adjustable trade-off among all aforesaid factors.

Index Terms—Plagiarism, copy-paste detection, chunking, matrix Bloom filter, hash function, cosine similarity measure.

I. INTRODUCTION

Plagiarism is the use of someone else’s findings as your own without any reference to original source. Unfortunately, such cheating has become an exceedingly easy and prevalent exercise between some researchers, who want to submit their thesis or work to colleges, conferences and other events. This is because of easy access to a enormous number of works via web [12]. There are different plagiarism methods such as syntactic plagiarism and semantic plagiarism. In syntactic plagiarism such as copy-paste method, one copies whole or partial of the work and pastes it into his/her own work and in later one, the plagiarizer steals the content of the work by utilizing different words. Furthermore, plagiarism can be occurred in various areas such as source code area [11], image area [33], video area [26] or text area. The scope of this work is to detect copy-paste plagiarism in the digital documents included in the libraries of colleges and conferences. Notice that we use the copy-paste term for both completely and partially similar documents. However, the degree of copy-paste detected by the system may be different for various documents. The main cause of selecting this area is the simplicity and so the prevalence of copy-paste process between students and researchers, according to many reports about this. There are many proposed methods trying to estimate the degree of similarity between two documents or a queried document against a database of documents [14], [25], [30]. These methods commonly consist of two steps: in the first step, the document is chunked into many sub-strings, and then in the second step, these chunks stored in a database or hashed values generated by some algorithms such as MD5 are compared to each other to find the fingerprints shared by two files. Also there are various chunking methods each of which utilizes the special length of chunks [19], [16]. For example, the SCAM system [24] tests several types of chunking size such as one word, ten words and sentences. On the other hand, more common similarity metrics are cosine function and dot product of the word frequency vector or asymmetric, symmetric and global similarity metrics mentioned in [16]. Most current methods proposed to detect similar documents, only focus on estimating the degree of similarity of two documents with a high accuracy. In other words, some important factors such as the speed of querying operation and memory usage to save chunks or fingerprints have not been sufficiently taken into account by these approaches. Moreover, in today information world, there are many applications in which the security of the file content should be ensured while performing copy detection process. As an instance, we assume two entities such as two conferences each of which wants to know whether a given submitted paper has been sent to other one without revealing the content of paper to other entity [31]. Therefore, a copy detection system should pay attention to this factor besides other mentioned factors. Since current methods have not designed to provide content security factor, they can not be applied in such applications. In addition, we believe that there is another new factor, called adjustability of the copy detection system, which is required to be considered in some applications. In this case, the relevant application can specify the important degree of each factor according to
its requirements. This new factor has not been considered in current proposed methods. To take into account all existing factors, in this paper we introduce a multi-purpose copy-paste detection architecture based on matrix Bloom filter which can consider all above factors according to the requirements of the relevant application. Our proposed approach provides some improvements on current methods, including enhancing the speed of estimation operation up to 2.7 times compared to the cosine similarity measure utilized by some current methods, eliminating the memory required to save all documents in MBF and ensuring the security of document content at the cost of tiny decreasing of the estimation accuracy. Moreover, it can give an adjustable property by which the relevant application can adjust the system factors according to its demands. A data structure, called Bloom filter, forms the main framework of the system. Bloom Filter (BF) is a space-efficient and randomized data structure for representing a set of elements, which was proposed in 1970’s by Burton Bloom, in order to verify the membership of a given element [4]. The matrix Bloom filter is defined as an extension of Bloom filter in which each entry occupies only one bit of memory, namely it can be set or reset. Each row of the matrix Bloom filter acts as an $m$-bits Bloom filter which is allocated to each document of database. Therefore, all documents needed to be maintained in database are stored in the matrix Bloom filter to reach above improvements. By using matrix Bloom filter, the ability to update the database and retrieve a special document is provided. In this architecture, the documents are chunked into the sub-strings with special length and hashed into the matrix Bloom filter. Each queried document is processed in a same way and compared with relevant Bloom filter by using a simple bitwise AND operation between their Bloom filters. Our approach has several main characteristics such as:

- Due to use of bitwise AND operation, the degree of similarity of two files can be estimated faster than previous methods that use other common similarity measures such as cosine similarity measure. Moreover, the query speed of our system is independent of the number of documents stored in database.
- The memory required to maintain a document in database is less than that of other approaches due to the bit structure of Bloom filters.
- It can inherently ensure the security of the content of files due to utilize several one-way hash functions for hashing chunks as well as saving this hashed values in a bit-array.
- It can provide some arbitrary options for the application so that it is possible to enhance the security of copy detection process at the cost of enhancing some more collisions or to enhance the accuracy of the system at the cost of consuming some additional memory. Therefore, the relevant application can adjust three mentioned factors according to its demands and interests.

The rest of the paper is organized as follows. In the next Section, we briefly discuss current copy detection approaches and related researches. In Section III, we review the concept of Bloom filter and the mathematical basic behind it. In Section IV, we describe our proposed copy-paste detection architecture. In Section V, we present our experimental results and analyze the performance of the system in terms of query speed, memory consumption and content security. Finally, in Section VI, we give a conclusion and propose some future works.

II. Related Work

In this section, we describe previous researches related to the work presented in this paper. The idea of detecting similar documents was introduced in [22]. Two documents are similar if the number of common fingerprints is greater than some pre-defined threshold. The basic idea of COPS [27] is similar to [22], but COPS uses a hash function to generate fingerprints of chunks of the document. It compares hash values of queried document with all chunks of other documents stored in database. In this case, the hash function produces a large number of collisions. The COPS system only can detect similarity in documents which have at least 2% overlap. Authors, in [3], aim to reduce the complexity of the system. At first, only some features of two documents are tested. If the primary conditions are yielded, the exact matching is performed. The idea behind SPLaT is to detect similarity at the sentence level [5]. To do this, it calculates the multiplicity of common words in two documents. Also, winnowing techniques has been proposed in [30] to detect similar documents in arXiv [10]. In [32], authors presented a method to succinctly representing a document to detect similar documents. The SCAM system [23] uses information retrieval techniques to perform word-based copy detection, but it is geared towards small documents. The similarity is defined in terms of closeness between terms and a subset measure is used to compare a pair of documents. The CDSDG system [13] uses the vector model to detect similarity. What is more, digital watermarking refers to the technique of attaching additional information to digital documents. It uses techniques such as line-shift coding, word-shift coding to place additional information in a document [15], [29], [21]. But, this method does not work well when the formatting of the document is changed. The method proposed in [8] is a sentence-based copy detection technique that can only detect same sentences. In general, most of these methods focus on one factor, namely estimation accuracy. In other words, there is no sufficient attention to other important factors such as the speed of estimating similarity degree of two documents and memory required to maintain documents in database. On the other hand, there are some new applications in which the content of documents should be concealed when performing copy detection process. It is clear that the current methods are not suitable to be applied in such applications. Our proposed approach can efficiently address these drawbacks. It can enhance the speed of estimating similarity degree between queried document and documents stored in database and reduce memory required to maintain all documents in database. In addition, all mentioned
factors can be adjusted according to the application demands due to the adjustable property of our proposed approach.

III. PRINCIPLE OF BLOOM FILTER

A Bloom filter is a simple, memory- and time-efficient randomized data structure for succinctly representing a set of elements and supporting approximate set membership queries. It was introduced by Burton Bloom in 1970 [4]. Initially, Bloom filter was applied to the database application, spell checkers and file operations [1]. In recent years, Bloom filters have been increasingly used in networking applications, including resource routing, web caching and etc [17], [1]. The basic Bloom filter has been extended in many ways to become appropriate for utilizing in specific applications. The memory efficiency gains are yielded at the cost of generating a low rate of false positives (nonmember elements have a small probability of being declared as a member) in membership queries but have no false negatives (member elements are not declared as a member of set). However, there are many network applications in which this tiny error rate is negligible in comparison with its significant benefits such as space and time efficiency. The BF uses an $m$-bits array $A$, initially all bits are set to 0, for succinct representing the data set $S = \{x_1, x_2, ..., x_n\}$ with at most $n$ elements. A Bloom filter uses $k$ independent hash functions $h_1, ..., h_k$, such as SHA-1, MD5 and $H_3$ [20], uniformly mapping each element of data set $S$ to a random number over the range $\{0, ..., m-1\}$. Constructing the Bloom filter consists of two phases: programming phase and querying phase [4], [1]. In programming phase, each element is hashed by $k$ independent hash functions. Then, all bits $A[h_i(x)]$ are set to 1 for $1 \leq i \leq k$. If a position in the Bloom filter array has set to 1 by inserting other elements, it remains unchanged in the next operations. In querying phase, $y$ is hashed by $k$ same hash functions for detecting whether an element $y$ is a member of $S$ or not. Then, if all these $k$ addresses in the Bloom filter are 1, we can conclude $y$ is in $S$. However, with a tiny probability of false positive, this conclusion may be wrong. That is, one or more positions have been set by inserting other elements of the set. If at least one of the $k$ positions is not set to 1, we can surely conclude $y$ is not in $S$. There is a trade-off between the probability of false positive $fp$ and the length $m$ of the Bloom filter array [4], [1]. It has been proven that the probability of false positive is equal to:

$$fp = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)$$

Now it is clear that the optimal number of hash functions $k$, minimizing $fp$, can lightly found by taking the derivative of above equation [4], [1]. Therefore:

$$k = \frac{m}{n}ln(2)$$

In recent decade, BF has been modified and improved from different aspects for a variety of specific problems. Some of these variations include counting Bloom filter [17], compressed Bloom filter [18], dynamic Bloom filter [9], spectral Bloom filter [28] and space-code Bloom filter [2].

IV. ADJUSTABLE COPY- PASTE DETECTION SYSTEM BASED ON MBF

In this section, we present an innovative adjustable copy-paste detection system based on matrix Bloom filter which benefits the idea behind Bloom filter in order to provide an adjustable system according to the requirements of the applications.

A. Matrix Bloom filter

Matrix Bloom filter is an extended variant of normal Bloom filter in which each individual row acts as a normal Bloom filter. Our motivation of introducing matrix Bloom filter is that there are some drawbacks with normal Bloom filters when utilizing it in designing our copy-paste detection system. First, normal Bloom filter can not support retrieving operation of a special document from Bloom filter to do exact estimation. This is because Bloom filter uses one-way hash functions to store elements and there is no capability to refer to a special element in the Bloom filter. While it is a simple operation to retrieve any special document from matrix Bloom filter because there is an individual row for each document stored in MBF. Second, when the normal Bloom filter satiates, it is not simple to update Bloom filter when adding any new document into the Bloom filter in the future. The only way to do this is redesigning Bloom filter in a bigger size in order to avoid more false positives. It is clear that this approach is not efficient at all. In contrast, updating matrix Bloom filter is a very simple operation. Only operation needed to be done is that for each new inserted document a new row is appended to the matrix Bloom filter in order to maintain this new document. Third, if we utilize normal Bloom filter in our copy detection approach, it is not possible to find out the similarity degree of the queried document against to the stored documents. In contrast, by using matrix Bloom filter, we can compare the Bloom filter related to the queried document with the Bloom filter related to each one of $N$ documents, separately. Matrix Bloom filter has all characteristics of a normal matrix except that each entry of matrix Bloom filter only occupies a single bit of memory. In other words, each entry of this matrix can only be set or reset. Similar to the normal Bloom filter, all entries of matrix Bloom filter are initialized by 0. Each row of the matrix Bloom filter is associated to each one of $N$ documents that must be stored in database. Therefore, the number of rows of matrix is equal to the number of documents. The number of columns in matrix Bloom filter depends on the number of bits that we want to allocate to store each document. The length of rows of matrix Bloom filter is called $sizeBF$ in this paper. The $sizeBF$ is one of the adjustable factors of our approach.

B. Our MBF-based copy detection system

In this section, we describe our proposed MBF-based approach to detect similar documents. Our main goal to utilize
Bloom filter as the core of our design is the simplicity as well as significant properties of this data structure such as its bitwise architecture. As most of existing copy detection systems, our design uses chunking methods to split documents into sub-strings, but the way of saving generated chunks in the database differs from that of other methods. In this case, neither the real chunks nor the hash values of these chunks are saved in database. The main core of our system is Matrix Bloom Filter, which each cell of this matrix can be only set or reset. In our design, this MBF is utilized to maintain all documents in database. Therefore, all common operations such as document retrieving, database updating and estimating similarity on the queried document are performed in the MBF. Each row of MBF is assigned to each document that should be saved in database. The general view of our architecture is depicted in Figure 1.

Fig. 1. Copy detection process of our MBF-based system.

Chinking Unit (CU) is responsible for splitting each arrived document into its sub-strings or chunks. Notice that the same chunking policy is used to extracting features of both N documents that should be saved in MBF and the queried document. At first, in order to create MBF containing all N documents, these N documents are passed through chunking unit one by one. The output of chunking unit is a set containing all chunks of relevant document. There are various chunking methods that can be used according to relevant application [19], [6]. In this paper, we intend to extract four types of chunks from the document and do copy detection operation based on each of them. These four chunking policies include 15-character strings, 20-character strings, 40-character strings and full sentence. By this, we want to show that our system can work independent of the style of chunking, called chstyle and can be a supplement system for current valid chunking methods. In the case of using full sentence as a chunk, we can use related algorithms such as campbell algorithm [7] to extract the set of sentences of the document. As most of existing systems, we assume that the content of the document has been converted to a plain text file by relevant tools and algorithms. The output set, called chset, consists of all chunks extracted from the document by one of mentioned methods. Then, each element n, of chset (n ∈ chset), i.e. each chunk, is utilized as an input for hashing unit (HU) so that each chunk included in the set is hashed by k hash functions. For each individual chunk in chset, hashing unit generates k positions in relevant row of matrix Bloom filter. Then, all these k positions in MBF are set to 1. This process is repeated for all documents that must be inserted into MBF. There are several different types of hash functions to apply in our approach such as SHA-1, MD5 and etc. However, in this paper we utilize a class of universal hashing functions, called H3 [20]. This is because the hashing functions from the class H3 are linear transformations. Therefore, we infer that by choosing at random from the class H3, theoretically predicted performance of hashing schemes can be achieved in our approach [20]. Let Q indicates the set of all possible i × j Boolean matrices. For a given q ∈ Q, let q(k) be the bit string which is the kth row of the matrix q, and let x(k) denotes the kth bit of x. The hashing function of hq(x) : A → B is defined as,

\[
h_q(x) = x(1) \cdot q(1) \oplus x(2) \cdot q(2) \oplus ... \oplus x(i) \cdot q(i)
\]

Where \( \cdot \) denotes the binary AND operation and \( \oplus \) the exclusive OR operation. The class H3 is the set \( \{ h_q \mid q \in Q \} \). The hashing function from this class can be easily implemented in hardware. In our approach, we generate several 256 × 32 Boolean matrix according to the value of k that we aim to use in our approach. In this paper, we examine the effects of different values of k on performance of our copy-paste detection approach. Each chunk as a bit string is fed into hashing unit resulting in generating k integer numbers in the range \{0,...,sizeBF − 1\}. In general, our copy-paste detection system consists of two phases: inserting phase and querying phase. In the first phase, each document that should be saved is split into its set of chunks, chset. These chunks then are hashed by k predefined hash functions embedded in Hashing Unit (HU). Figure 2 depicts the inserting algorithm. The output of each function is an index number in range \{0,...,sizeBF − 1\}, where sizeBF denotes the number of MBF’s columns. Then, the corresponding cells of the row of MBF, which has been associated to this document, are set. This process is used for all documents that should be saved in MBF. In the second phase, the queried document that we aim to estimate the degree of its similarity with one or all documents stored in MBF, is split into its chunks by CU, same as inserting phase. All elements of set generated by CU are hashed by the same k hash functions embedded in HU, leading to create a bit-array of the queried document. After this point, unlike existing methods which use similarity measures such as cosine similarity or jaccard
similarity, our system uses only a bit by bit AND operation between two BF of each document pairs to figure out the degree of copy-paste between them. The degree of similarity is detected according to the number of bits set to 1 in resultant array. Figure 3 depicts the querying algorithm. Where is the bit-array resulted from AND operation between two Bloom filters of two documents. The variable indicates the number of positions set to 1 in BF. The variable indicates the number of chunks of the document and is the rate of similarity of two documents, i.e. queried document and document stored in row of MBF. In this case, the threshold value can be defined to indicate the similar files. Our observations show that if the degree of similarity of two files is estimated about 90%, for instance, so they can be deduced as similar with a high probability. It is obvious that if the degree of similarity is about 10%, for instance, they can be deduced as dissimilar.

It is obvious that our system is very simple to implement and also it is fast because of using bitwise operation. Notice that due to the inherent property of Bloom filters in generating some false positives, it is possible that dissimilar files are introduced as similar with the similarity degree of between 6%-10% as we present in the next section. Moreover, the presence of some share phrases between all documents can enhance this amount of false estimation. Some of these phrases are:

The rest of the paper is organized as follows. Finally, section 5 concludes this paper. The following figure shows this architecture.

In the case of our application, the false positive term is defined as the amount of false similarity estimated by our system for two dissimilar documents.

V. EXPERIMENTAL RESULTS

In this section we discuss the performance of our approach by presenting the results of our experiments for different factors. As it was said before, this system provides a tradeoff between several given factors according to the requirements of the related application. Moreover, we analyze three improvements introduced by our proposed approach, including the speed of estimating process, memory saving and content security.

A. Performance of our adjustable MBF-based copy detection system

To see the outputs of our system, we show the performance of our system to detect the degree of copy-paste between the queried document and all documents existed in MBF for different factors. These factors include the length of each row assigned to each document $sizeBF$, the number of hash functions $k$, and the style of document chunking $chstyle$. Notice that we tested the proposed system over a digital library containing about 10000 documents but due to space limitation, we present the output of the system only for a few numbers of documents stored in MBF, here 20 documents. Also, the average numbers of pages of each document is assumed to be about 6-8 pages. Our system is tested on documents written in English language. Here, $QD$ and $D_i$ denote the queried document and stored documents in MBF, respectively.

1) Effect of matrix Bloom filter size: To verify the effect of different lengths of bit-array assigned to each document, i.e. the number of columns of MBF $sizeBF$, we demonstrate the output of system for multiple values of $sizeBF$ in Figure 4. In this case, we let other factors remain as it was. Due to page limitation, we only test four values of $sizeBF$, including 3000, 5000, 7000 and 10000 bits. It is obvious that by increasing the value of $sizeBF$, the accuracy of our system is increased, too. In Figure 4, the similarity degree of the queried document $QD$ against all 20 documents stored in MBF has shown for four different values of $sizeBF$.

![Fig. 4. Similarity degree between the queried document QD and documents stored in MBF for k = 4, chstyle = sentence and four different values of sizeBF.](image)

![Fig. 5. The average effect of sizeBF on the estimation accuracy.](image)
Other two factors have set fixed, i.e. \( k = 4 \) and \( \text{chstyle} = \text{sentence} \). The accuracy of the system for \( \text{sizeBF} = 10000 \) is about 99% for two completely similar documents. The system introduces some similarity for two dissimilar documents. For example, we expect that the similarity degree of two documents \( QD \) and \( D_1 \) to be small, while according to Figure 4, this value is about 9%. This is because of two reasons: first, the collisions generated by \( k \) hash functions lead to enhance the multiplicity of 1’s in bit-array and second there is the probability of occurring some value of random similarity between two different documents. Figure 4 presents that this false similarity can be reduced by increasing the value of \( \text{sizeBF} \). This demonstrates that our system can be adjusted based on the defined policies in the relevant application.

In such an application, a threshold value can be defined while specifying similar documents. However, these collisions always are not so bad because of providing some security gains, as we will discuss in the section V-B. We performed this examine for other different values of \( \text{sizeBF} \) and found out the same conclusions. Figure 5 depicts the average accuracy of our system for all documents stored in MBF and different values of \( \text{sizeBF} \) values. Notice that the estimation accuracy of the system has remained fixed for \( \text{sizeBF} \) values more than 20000.

2) Effect of different numbers of hash functions: Hash functions inherently introduce some false positives in each application that exploit them. In the context of this application, the false positive is defined as the amount of false similarity estimated for two dissimilar documents. Figure 6 depicts the effect of the number of hash functions on the accuracy of similarity estimation.

As we can observe, if the number of hash functions is increased, the accuracy of the similarity estimation for similar documents is reduced somewhat. On the other hand, the degree of the false similarity increases. For example, in the case of \( k = 4 \), the amount of false similarity for \( D_1 \) is equal to 8%. While this amount for \( k = 7 \) is about 12%. Notice that we expected that the rate of similarity for two documents \( QD \) and \( D_1 \) to be zero. Also, in the case of document \( D_{17} \), for instance, the accuracy of estimation has reduced about 2% when increasing the amount of \( k \) from 4 to 7. Our experiments show that the number of three or four hash functions can be sufficient for this application because this number of \( k \) can ensure the privacy of content of documents. Figure 7 presents the effect of various number of \( k \) on increasing false similarity degree of two dissimilar documents.

3) Effect of different chunking methods: Indeed, the method used to chunk a document has a great influence on the performance of each copy detection system. A good chunking method can enhance the accuracy of a copy detection system. Although, accuracy is not the only factor of evaluating our multi-purpose system and some other goals should be considered for special applications, but a powerful chunking method can enhance the performance of our system, too. In other words, our system can be applied on other chunking methods so that it can be an adaptable storing and querying framework for any good chunking method. Here, to show the performance of our system, we have used multiple different chunking styles. The functionality of the system for these four styles has been shown in Figure 8. The figure shows that with increasing the size of chunks the accuracy of the estimation process is slightly getting better. Notice that utilizing full sentence as the input of our system results in best accuracy. Our additional experiments show same conclusions.

B. Overall result discussion

In this section, we analyze three main improvements provided by our approach, including enhancing the speed of esti-
mating process, reducing memory requirements and providing security of document content.

1) Analyzing the speed of querying operation: Querying operation means the process of finding the degree of similarity between the queried document and each one of $N$ documents stored in database. The speed of this process is one of the adjustable factors of our system. We believe that the query speed of proposed approach is better than that of other methods due to utilize bitwise AND operation of two bit-arrays related to two compared document. To prove this, we compared our proposed approach with one of the most popular similarity measures called cosine measure. The methods based on this measure usually save a dictionary of all chunks or terms of documents in database and count the frequency of terms to generate the vector space of document. Then, the cosine between these two vectors shows the degree of similarity between documents. The result of our experiment is depicted in Figure 9.

![Fig. 9. The speedup of our approach in comparison with cosine similarity measure for different number of documents stored in database.](image)

This figure demonstrates the speedup of our approach in comparison with cosine similarity measure for different number of documents stored in database.

We can see that with increasing the number of documents in database the ratio of query speed of our system is getting better compared with cosine measure. Because, with increasing the number of documents and then the size of dictionary, the speed of calculating cosine value increases, too. Therefore, we can conclude that the query speed in our approach is somewhat independent of size of database. Notice that based on the adjustability property of our approach, the time required for estimating similarity can be decreased (increased) at the cost of increasing (decreasing) sizeBF.

2) Analyzing the space consumption: As we said before, the existing methods to address copy detection problem are usually based on generating all chunks of documents and estimating the degree of similarity by using a similarity measure such as cosine. Most of these works directly use the set of chunks or a hash value of them to detect similar documents. Some other works use the frequency of occurring each of these chunks to form the vector space of documents. Unfortunately, there is not obvious statement about the way of how saving these chunks or fingerprints in database. However, we believe that directly saving the fingerprints or chunks in database leads to consume high amount of space. Also, in the context of frequency-based methods, the storage requirements to store a dictionary of all terms can be very high. In our proposed approach, the size of bit-array required for saving a document of 6-8 pages range is averagely 5000 bits or about 0.6 kbytes. Of course, this value is appropriate for detecting the similarity with the accuracy of between 96%-98% as we aimed in this work. There are some techniques such as sampling techniques consuming lesser memory but our system can use the sampling techniques to decrease memory usage, too. In this case, only the special number of chunks are stored in MBF. Similar to other factors of the system, the allocated storage can be adjusted according to the requirements of the relevant application.

3) Analyzing the content security: All Existing copy detection methods are based on this assumption that the privacy of the document content is not an important issue. Therefore, there is no any attention to pervers the content of documents. However, these methods may be indirectly achieve a very low level of privacy due to use of one hashing function by some of this work. But this level of security is not sufficient in some application in which the security of document content is a main factor. Most conferences and journals do not allow double submission. Similar papers detection methods should be used by conferences to compare submitted papers without disclosing the content of papers. To deal with such a problem we need similar detection methods which ensure the privacy of document content along with other main factors. Our approach is one of the first approaches which can be used to apply in such an application according to the required level of the security of content. The considerable note is that the $k$ hash functions used in our approach provide the required level of privacy without using more encryption techniques. This consideration can be discussed from two aspects: first, in our approach we used $k$ hash functions of $H_3$ class, leading to generate $k$ random numbers. Because of using $k$ one-way hash functions, it is impossible to retrieve the original string values from the Bloom filter. Second, bitwise structure of our approach naturally enhances the level of security because the outputs of hashing functions are used to set the corresponding bits in Bloom filters. Therefore, what we have in the end of hashing process is only a bit-array of 1’s and 0’s so that information extracting from these binary strings is impossible. Moreover, by increasing the number of $k$, the security level of content is increased at the cost of enhancing some false similarity degree. This demonstrates the adjustable capability of our approach, too. Our approach can be utilized by such conferences to detect double submitted papers. In this case, the conference only needs to send the Bloom filter related to the paper to other conference in order to recognize similar papers.

VI. CONCLUSIONS AND FUTURE WORKS

We have proposed an innovative copy-paste detection system which provides a trade-off among accuracy, speed, space
and security factors of the system. This idea comes into being from the belief that in many real applications, it is not necessity to detect the degree of similarity even 100% of accuracy. Moreover, in many applications such system should takes into consideration the privacy of content during detecting similarity. The core of our system is a Bloom filter-based architecture, called matrix Bloom filter, which can be adjusted to allow this trade-off according to the requirements of relevant application. We believe that our system has two new properties as follows: first, this is the first work using the idea behind Bloom filters to address copy detection problem while ensuring the privacy of document content. Second, this is the first work aiming to introduce an adjustable system to take into account other important factors such as speed, memory and security besides main goal, i.e. similarity estimation. We demonstrated that by allocating only about 0.6 kbytes of space to save a document with the number of pages in the range from 6 to 8, it can be sufficient to detect the degree of similarity between 96%–98%. The results of our experiments also show that the query speed of our bitwise AND-based similarity measure is better than previous applied measures up to 2.7 times for about 10000 documents. Moreover, $k$ hash functions applied by system provide a high level of security of content which should be ensured in some applications. There are several questions which can be addressed as future works. First, we believe that our idea can be extended to address other types of plagiarism. Second, this system can be tested to recognize similar documents with larger size. Third, we believe that our system can be extended to provide the higher levels of content security in special applications by adding other security-related techniques.

REFERENCES


