Journal of Computers Vol. 30 No. 2, 2019, pp. 150-157
doi:10.3966/199115992019043002013

A Method of Piecewise Hash for Fuzzy Hashing

Tian-zhou Li[1*], Bo Shen[1,2], Kun Mi[3], Yi-Chih Kao[4], Yong Cui[5]

[1] School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China
17120189@bjtu.edu.cn

[2] Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing 100044, China
bshen@bjtu.edu.cn

[3] Beijing Thunisoft Company, Beijing 100084, China
kunmi70@sina.com

[4] Information Technology Service Center, National Chiao Tung University, Taiwan
ykao@mail.nctu.edu.tw

[5] Beijing Thunisoft Company, Beijing 100084, China
cuiyong@thunisoft.com

Received 6 December 2018; Revised 6 February 2019; Accepted 6 March 2019

Abstract. The fuzzy hash algorithm was originally applied to computer forensics and then widely used in the fields of malicious code detection, homologous similar file detection and electronic data forensics. The primary process of fuzzy hash algorithm divides the detected file into some fragments with fixed length and calculates the hash value of each fragment by rolling hash compose a fingerprint of the file. The length of fragment is associated with the file size, which means for a large file the fragment will be long and for a short file it will be short. This paper introduces a new fragmenting rule and a method of preprocessing files to overcome the weakness and enhance the efficiency of processing small files. Experimental results indicate that the pro-posed method has better performance.

Keywords: file fragmentation, fuzzy hash, weak hash

1 Introduction

With the advent of the information age, information retrieval, and information security testing have played an essential role in fields of communication. For virus prevention, malicious code detection, and data leakage prevention. It is necessary to diagnose files' homologies by combining whitelists and blacklists. These problems can be attributed essentially to the nearest neighbor search problem in high dimensional space. Therefore, using a hash algorithm will improve the process of diagnosing more quickly and efficiently. Essentially, these problems can be attributed to the nearest neighbor search problem in high-dimensional space. The application of hash algorithm in solving these problems can significantly improve the efficiency of file comparison, reduce the required storage capacity and reduce the complexity of comparison operations.

To be more specific, hash algorithm takes an arbitrary length of data as input and shrinks it down into a small and fixed hash value. Furthermore, this sort of symbol is related to every byte; and it is difficult to find reverse law to figure out inputs through outputs. That suggests when the original file changes, its hash value will also change. This is widely used for quick searching of the original information, and password comparison. According to different applications, hashing algorithms can be divided into index hashing, encryption hashing and locally sensitive hashing.

[*] Corresponding Author

As an efficient approximate nearest neighbor search algorithm, locally sensitive hashing can effectively overcome the dimensionality disaster in high-dimensional data search [1]. It has achieved many results in plagiarism detection, classified file discovery, web page duplication, malicious code detection and other applications. When comparing file similarity in these applications, it is expected that the same source files with similar content but not identical data can have similar or identical hash values. Because the comparison process is very similar to the qualitative comparison process of fuzzy logic, the hash algorithm with this characteristic is also called the fuzzy hash.

A file is intentionally or unintentionally changed, for example, the author changes the text content, the malicious code changes automatically, the transmission error, the disk storage error, etc. How to determine the similarity of documents, and whether they are homologous are crucial problems in all fields. In order to solve these problems quickly and efficiently, in 2006, Jesse Kornblum proposed the CTPH [2] (Context Triggered Piecewise Hash) algorithm. The CTPH algorithm first-ly uses a weak hash to compute and segment the local content of the file, under certain circumstances, and then computes a hash value for each piece of the file using a strong hash. Combining some portions of hash values and conditions of segmentation construct a result of fuzzy hashing.

CTPH is the first file comparison algorithm to achieve implementation of fuzzy hashing. This algorithm could figure out the similar relationship between the original files and its partial changes, including modifying, adding, and deleting multiple contents. The fuzzy hash was originally applied to computer forensics [3]. Immediately, the anti-virus field discovered its beauty and tried to use it for the detection of malicious code [4-6]. Nonetheless, the result and effect of comparison among relatively small files or those files that are specially processed are unsatisfactory. In this paper, we aim to solve problems above by extracting fragmentation conditions and preprocessing files by special information extraction [7].

2 Locality-Sensitive Hashing

LSH (Locality Sensitive Hashing) refers to the use of a specific hash algorithm to locate high-dimensional data into low-dimensional space to search similar sets of data rapidly with the higher possibility [8]. As a highly efficient approximation nearest neighbor search algorithm, local sensitive hash can effectively overcome dimensional disasters in high-dimensional data search [9]. Since Locality-Sensitive hashes has the similar processes compared to the definitive comparison of fuzzy logic, LSH could also called fuzzy hashes.

In many areas, the data we need to deal with are often massive and have high dimensions. Comparing the similarity of data needs to find the nearest neighbors in high-dimensional space. However, for the high-dimensional space nearest neighbor search problem with data set size N and dimension $D \geq 3$, there is no nearest neighbor search algorithm with space complexity of N at linear multiple level and time complexity of N at logarithmic level, so many researchers turn to search for approximate solution of nearest neighbor search problem [10]. INDYK et al. proposed a locally sensitive hashing algorithm, and proved that after the pretreatment process with time complexity of $dN^{0(1)}$, the searching point's ε-nearest neighbor can be obtained with approximate linear time complexity $O(dN^{\frac{1}{1+\varepsilon}})$. [11]

The basic idea of LSH: After the two adjacent data points in the original data space are transformed by the same mapping or projection, the probability that the two data points are still adjacent in the new data space is very high, while the probability that the non-adjacent data points are mapped to the adjacent range is very small. If we do some hash mapping on the raw data, we hope that the two adjacent data can be hashed into the same room with the same room number. After hash mapping all the data in the original data set, we get a hash table. These original data sets are scattered into the room of the hash table. Each room will fall into some raw data and belong to the same room. The data is likely to be adjacent, and of course there are non-adjacent data that is hashed into the same room. Therefore, if we can find such hash functions, so that after the hash map transformation of them, the adjacent data in the original room falls into the same space, then it is easy to make a neighbor search in the data set. However, we only need to hash the query data to get its room number, then take out all the data in the room corresponding to the room number, and then perform linear matching to find the data adjacent to the query data.

In other words, we divide the original data set into several sub-sets by hash function mapping transformation, and the data in each sub-set are adjacent and the number of elements in the sub-set is small. Therefore, the problem of finding adjacent elements in a large set is transformed into the problem of finding adjacent elements in a small set, which obviously reduces the computational complexity.

If a family of functions F : x, y is any two data points in the high-dimensional space, and its distance measurement function is $d(x, y)$, which satisfies any hash function on F :

$$\begin{cases} d(x, y) \le d_1, \text{then } p(h(x) = h(y)) \ge p_1. \\ d(x, y) \ge d_2, \text{then } p(h(x) = h(y)) \le p_2. \end{cases} \quad \textbf{(1)}$$

Where $p_1 > p_2, d_1 < d_2$, function h is a hash function with local sensitivity, then the function family $F(d_1, d_2, p_1, p_2)$ is sensitive [12].

Locally sensitive hashes maximize data similarity.

2.1 Implementation of Fuzzy Hash Algorithm CTPH

The fuzzy hash uses the idea of piecewise to maintain data similarity. Before this, Nick Harbour proposed a piecewise hashing and implemented it in dcfldd [13]. The strategy is very simple, that is, to segment every fixed length interval into pieces, calculate the hash value for each piece, and compare these hash values together for similarity. The local modifications only impact results of some individual fragment hashes, which ultimately enhances the efficiency of comparing similarities. However, it is catastrophic to add or delete bytes. In order to solve this issue, CTPH determine whether segment or not based on the characteristics of local data, instead of fixing the length of segmentation. That eventually only cause an influence of local piecewise method, when there are local changes which includes modification, inserting, and deleting, Meanwhile, it would extent to other pieces.

CTPH primarily uses a weak hash algorithm, with high conflict probability, to trigger piecewise detection. And then, he segmentation position is determined by the scrolling block by rolling hashes. Moreover, each length of pieces is adjusted dynamically by the sizes of scrolling blocks. Then using the strong hash algorithm to calculate each fragment to acquire the hash value. In order to improve efficiency and save storage space, CTPH compress the obtained hash values, shrinks them from high-dimension to low-dimension, acquire a shorter value, and puts them together to form the file's fingerprint.

Regarding CTPH detection, it uses a weak hash algorithm inspired by Adler-32 [14]. To increase speed, CTPH uses a fixed-length scrolling block to calculate the text byte by byte. Mathematically, let input n characters, the character input in the ith time is b_i, when the size of the scroll window is s, when the pth character is sliced, the calculated hash value can be expressed as

$$r_p = F(b_p, b_{p-1}, b_{p-2}, ..., b_{p-s}). \quad \textbf{(2)}$$

Rolling the hash function F can eliminate the effect of one of them, so giving r_p calculates r_{p+1} by removing the effect of b_{p-s}, when the window moves backward by one bit,

$$r_{p+1} = F(b_{p+1}, b_p, b_{p-1}, ..., b_{(p-s)+1}).. \quad \textbf{(4)}$$

The CTPH condition is determined by the file length n, the scroll block length s, and the minimum piece length $b_{\min}$.

$$b_{init} = b_{\min} 2^{\left\lfloor \log_2 \frac{n}{\min} \right\rfloor} \quad \textbf{(5)}$$

When implementing modulo operation of r_p to b_{init}, resulting of b_{init-1}, it ought to be fragmented at this position. Furthermore, the size of the total number of fragments is adjusted by the size of b.

After the condition is triggered, the fragment is subjected to calculate by using a strong hash operation of FNV-1 [15]. As a result, a 32-bit hash value is obtained by only exacting and compressing the LS6B (Least significant six bits).

Finally, matching score of two fuzzy hash values is computed by the weighted average distance. First judge the change from s_1 to s_2, at least how many steps to operate (including insert, delete, modify, exchange), and then give a weight to different operations, add the results, that is the weighted edit distance. Next divide this distance by the sum of the lengths of s_1 and s_2 to make the absolute result a relative result and then map it to an integer value of 0-100.

CTPH believes that when the score is greater than zero, it indicates that the two files are similar. Especially the score is 100, the two files are exactly the same.

$$M = 100 - \left(\frac{100Se(s_1, s_2)}{64(l_1 + l_2)} \right) \tag{6}$$

Where $e(s_1, s_2)$ is expressed as the weighted average distance of two fuzzy hash values, l_1, l_2 are the length of s_1, s_2, and the default value of s is 64.

After the proposal, Jason Sherman developed the ssdeep [16] tool to achieve the idea of CTPH algorithm.

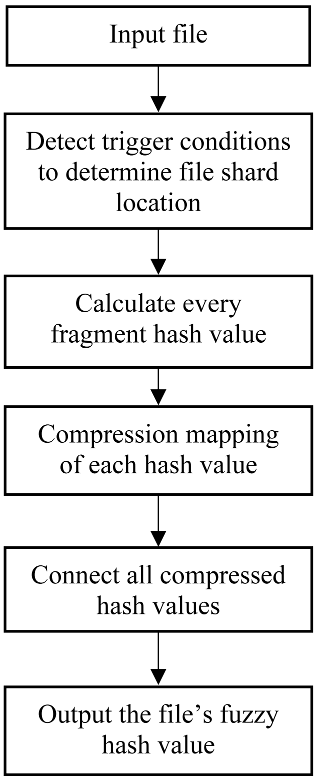
Input file
Detect trigger conditions to determine file shard location
Calculate every fragment hash value
Compression mapping of each hash value
Connect all compressed hash values
Output the file's fuzzy hash value

Fig. 1. Fuzzy hashing algorithm flow chart

2.2 Drawbacks of Fuzzy Hash

Comparing similar files can be achieved by calculating their fuzzy hash value. Nevertheless, it also arises the following two drawbacks when analyzing the hash values according to real experimental statistics.

Fixed-length piecewise allow files with different lengths to have equal-length hash values; however, comparing files with different lengths, it causes alignment errors. Moreover, when implementing longer file comparison, it is necessary to process adjustments on the segmentations for serval times [17], which eventually deteriorates and the accuracy of comparison.

When the size of input data is small specially less than 10KB, the algorithm is susceptible to unrelated characters and content. For instance, if you add symbols (such as spaces) to the content of the file, the fragmentation is affected accordingly.

When using weak hash to detect the trigger condition of fragmentation, the termination position of fragmentation only depends on the value of s strings in the scrolling window before the termination position, but has nothing to do with the characters outside the window, so it can not reflect all the file contents in the fragmentation.

The byte-by-byte scrolling operation is only applicable to English files. When calculating Chinese characters with 2~4 bytes each character, there will be a case where one Chinese character is divided into two different fragments.

3 Improvement Approach

This paper overcomes the above shortcomings by using preprocessing of files and special information extraction.

First, because of the effects of special characters and extraneous content, the file needs to be preprocessed by culling spaces, carriage returns, and irrelevant content; such as the "//" comment symbol and the following content while diagnosing malicious code.

Additionally, if the input file is small, the number of shards should not be limited. Thus, the previous fixed-length hash value transfers to variable-length hash value. On the other hand, changing the piecewise strategy and triggering only by special information also improve the process.

Last but not least, the FNV-1 is still applied to calculate the strong hash, as well as LS6B is implemented to obtain the fingerprint information, and the similarity is compared by the weighted average distance.

3.1 Special Information Trigger Fragmentation Strategy

In natural language, words are the least carrier of semantics. In this report, words are considered as the condition for the length of files. By extracting the keywords of the text, the fragmentation symbol is located to the keyword and triggers the frag-mentation.

In the text, the symbol can represent the pause or the end of a sentence, and the symbol has the same effect in the code. Therefore, some specific symbols can be used as the condition to trigger the fragmentation.

Different information can be established to trigger fragmentation according to the size of the comparison file. For example, when the input file is small, the segmentation can be triggered by a pause or a final symbol [18], such as ",", ".", "?", etc. Thus, the text can be compared to an exact sentence. When the input file is too large, the fragment is triggered by the number of occurrences of the keyword. To be more specific, a threshold s can be set. When the number of occurrences of the keyword in the scroll block $h \geqslant s$, the fragment is triggered. What is more, the first symbol representing the termination that appears after symbolling the keyword triggers segmentation condition.

3.2 Special Method

The file input is processed first, irrelevant content is removed, and then keywords are extracted. Since this paper is to test the text, TextRank [19] method is applied in key-word extraction. Implementing TextRank takes full advantage of context to extract keywords. When the key words are extracted, sub-slice and marker are added. This article is adding "." for the flag bit. Take "." for special information as a condition for piecewise. Finally, the segmented pieces can be computed by FNV-1 hash operation, following with LS6B calculation to construct a fingerprint information.

4 Specific Method

As shown in Fig. 2, the file input is processed first, irrelevant content is removed, and then keywords are extracted. Since this paper is to test the text, TextRank method is applied in keyword extraction. Implementing TextRank takes full advantage of context to extract keywords. When the key words are extracted, sub-slice and marker are added. This article is adding "." for the flag bit. Take "." for special information as a condition for piecewise. Finally, the segmented pieces can be computed by FNV-1 hash operation, following with LS6B calculation to construct a fingerprint information.

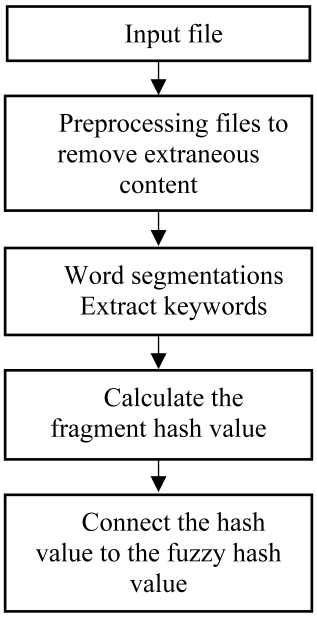
Input file
Preprocessing files to remove extraneous content
Word segmentations Extract keywords
Calculate the fragment hash value
Connect the hash value to the fuzzy hash value

Fig. 2. improved hashing algorithm flow chart

5 Experimental Results and Analysis

In the previous test of ssdeep, it was found that the performance was poor when the input file was small. Therefore, this paper randomly selects the worst-performing from 5KB to 8KB in the ssdeep test, randomly modifies the 1-100 words in the text, and calculates the fuzzy hash value to compare with the original file fuzzy hash value. The comparison is shown in Fig. 3 to Fig. 6.

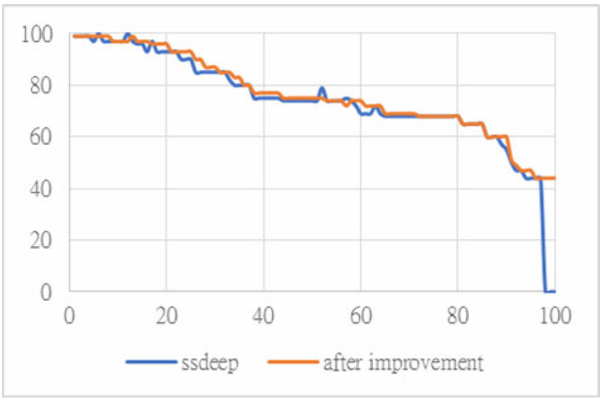

Fig. 3. Comparison of scores when input file 5KB size

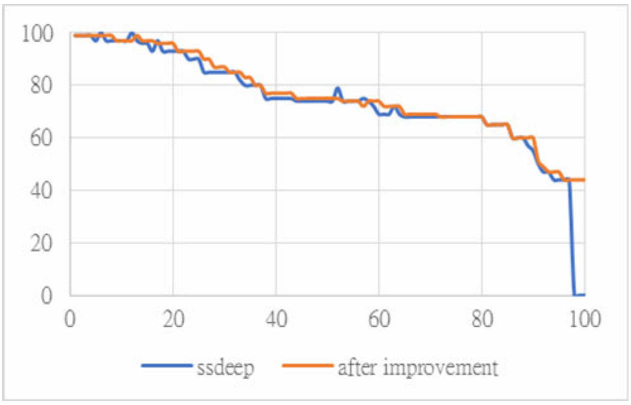

Fig. 4. Comparison of scores when input file 6KB size

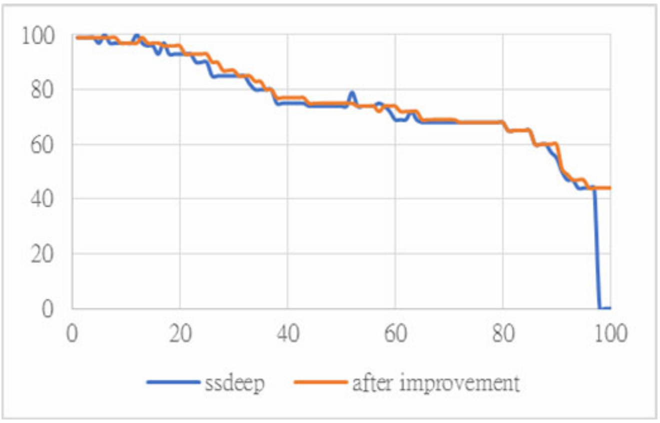

Fig. 5. Comparison of scores when input file 7KB size

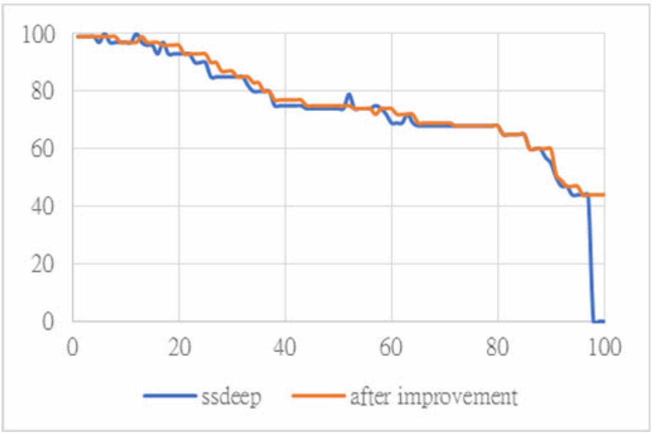

Fig. 6. Comparison of scores when input file 8KB size

It illustrates from the figure that when the modified place is greater than a certain value, ssdeep will determine that the two files are completely dissimilar. The reason for the analysis is that when the file is small, and place where the random modification is made everywhere. Although it is only the replacement of the vocabulary, this has affected the overall location of fragmentation. That ultimately affects the result of the judgment. Nonetheless, by using improvement approach, although the modification could affect the whole situation to some extent, those segmentation based on the special information can only be influenced by modifications where locates in the same place. Therefore, the performance will be improved to some extent without impacting overall segmentations.

6 Conclusion

This paper further enhances the fuzzy hashing method by extracting special information to trigger fragmentation rules and preprocessing the files. Moreover, it also improves the efficiency of the smaller files with poor performance under ssdeep. How to improve the process of large files is still a conception. The future work is to further refine the conception and figure out a way to apply files of various sizes in the near future.

Acknowledgements

This research was funded by the National Key Research and Development Program of China, grant number 2018YFC0831300, and the Fundamental Research Funds for the Central Universities, grant number 2017JBZ107.

References

[1] J. Zamora, M. Mendoza, A. Héctor, Hashing-based clustering in high dimensional data, Expert Systems with Applications 62(2016) 202-211.

[2] J. Kornblum, Identifying almost identical files using context triggered piecewise hash-ing, Digital Investigation 3(3)(2006) 91-97.

[3] B. Harald, B. Frank, Security aspects of piecewise hashing in computer forensics, in: Proc. 2011 Sixth International Conference on IT Security Incident Management and IT Forensics, 2011.

[4] R. Daniel, Automated malware similarity analysis, in: Proc. 2009 Black Hat 2009.

[5] A.P. Namanya, Q.K.A. Mirza, H. Almohannadi, J. Pagna-Disso, I. Awan, Detection of malicious portable executables using evidence combinational theory with fuzzy hashing IEEE, in: Proc. 2016 International Conference on Future Internet of Things and Cloud, 2016.

[6] Y. Li, S.C. Sundaramurthy, A.G. Bardas, X. Ou, D. Caragea, X. Hu, J. Jang, Experimental study of fuzzy hashing in malware clustering analysis, in: Proc. 2015 The Workshop on Cyber Security Experimentation & Test, 2015.

[7] M. Datar, N. Immorlica, P. Indyk, V.S. Mirrokni, Locality-sensitive hashing scheme based on p-stable distributions, in: Proc. 2004 Twentieth Symposium on Computational Geometry, 2004.

[8] P. Indyk, R. Motwani, Approximate nearest neighbor: towards removing the curse of dimensionality, Theory of Computing 604-613(11)(2012) 604-613.

[9] J. Zamora, M. Mendoza,A. Héctor, Hashing-based clustering in high dimensional data, Expert Systems with Applications 62(2016) 202-211.

[10] J.-F. Wang, Hash-based nearest neighbor search, [dissertation] Hefei, Anhui: University of Science and Technology of China, 2015.

[11] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proc. 30th Annual ACM Symposium on Theory of Computing, 1998.

[12] H.-Y. Di, J. Zhang, Y. Yu, L.-Y. Wang, Research on document comparison algorithm based on modiifed fuzzy hash, Information Network Security 2(11)(2016) 12-15.

[13] N. Harbour, dcfldd. < http://dcfldd.sourceforge.net/>, 2016.

[14] Wikipedia, Adler-32. < http://en.wikipedia.org/wiki/Adler-32>, 2018.

[15] Wikipedia, Fowler, Noll–Vo hash function. <https://en.wikipedia.org/wiki/Fowler%E2%80%93Noll%E2%80%93Vo_hash_function>, 2018.

[16] S. Jason, ssdeep. < http://ssdeep.sourceforge.net>, 2018.

[17] F. Breitinger, H. Baier, Performance issues about context-triggered piecewise hashing, in: Proc. 2011 International Conference on Digital Forensics and Cyber Crime, 2011.

[18] B. Say, An information-based approach to punctuation, in: Proc. 1997 Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence, 1997.

[19] R. Mihalcea, TextRank: bringing order into texts, Emnlp 32(2004) 404-411.