A Dynamic Locality Sensitive Hashing Algorithm for Efficient Security Applications

Mohammad Y. Khanafseh
malkhanafseh@birzeit.edu
Computer Science Department
Birzeit University
Ramallah, Palestine

Ola M. Surakhi
o.surakhi@aum.edu.jo
Computer Science Department
American University of Madaba
Madaba, Jordan

Abstract: The information retrieval domain deals with the retrieval of unstructured data such as text documents. Searching documents is a main component of the modern information retrieval system. Locality Sensitive Hashing (LSH) is one of the most popular methods used in searching for documents in a high-dimensional space. The main benefit of LSH is its theoretical guarantee of query accuracy in a multi-dimensional space. More enhancement can be achieved to LSH by adding a bit to its steps. In this paper, a new Dynamic Locality Sensitive Hashing (DLSH) algorithm is proposed as an improved version of the LSH algorithm, which relies on employing the hierarchal selection of LSH parameters (number of bands, number of shingles, and number of permutation lists) based on the similarity achieved by the algorithm to optimize searching accuracy and increasing its score. Using several tampered file structures, the technique was applied, and the performance is evaluated. In some circumstances, the accuracy of matching with DLSH exceeds 95% with the optimal parameter value selected for the number of bands, the number of shingles, and the number of permutations lists of the DLSH algorithm. The result makes DLSH algorithm suitable to be applied in many critical applications that depend on accurate searching such as forensics technology.

Keywords:
High Dimensional Data, Hash Function, Information Retrieval, Locality Sensitive Hashing, Nearest Neighbors Search, Similarity

1. Introduction

With the development of advanced digital technologies, many high-dimensional datasets grow exponentially. Managing these data sets requires a suitable dynamic mechanism that approximates the search results instead of searching for exact results [1]. The nearest neighbors method in high-dimensional space is effective in many important applications such as information retrieval [2], audio fingerprinting [3], biological and geological sciences [4], and more. The method searches for results that are close enough (not 100% of accuracy) to the required data. For a high number of dimensions, the nearest neighbors method suffers from a well-known problem, the curse of dimensionality where the performance decreases with the increase in the number of features [5].

Many hashing algorithms have been proposed and used for high-dimensional data such as dimensionality reduction, data clustering, and classification algorithms to increase the accuracy and searching speed of the nearest neighbors and overcome its limitations [6]. The Locality-Sensitive Hashing (LSH) algorithm [7] is an efficient algorithm for dimension reduction problems and data clustering [6]. The main concept of the LSH algorithm is to map the high-dimensionality data to lower-dimensionality data using a random hash function [8]. For each hash function, a set of data points are assigned to individual hash buckets so that the closer data points in the original high-dimensional data will be mapped to the same hash bucket in the low-dimensional data.

The LSH can be applied in many applications that depend on finding similarities between different data points such as information retrieval, data mining, classification problems, and more due to its theoretical guarantee of query accuracy. The LSH algorithm also uses a random hash function which makes it data-independent where the distribution of these data does not affect the generation of the hash function. Additionally, for real-time applications where new data are generated instantaneously, the hash function does not require to be changed during runtime.

Due to its importance in many applications, many variants of LSH have been proposed to improve its performance and enhance searching accuracy [9-13]. The large index size problem in the LSH algorithm has been solved by proposing many techniques such as

Manuscript received May 5, 2024
Manuscript revised May 20, 2024
https://doi.org/10.22937/IJCSNS.2024.24.5.9

collision counting and virtual rehashing [14, 15]. Using a small index guarantees fast query performance and accuracy.

In this paper, we propose a new extension of the LSH algorithm that is based on a hierarchy structure to increase the performance of the algorithm. The proposed algorithm, Dynamic Locality Sensitive Hashing (DLSH) Algorithm uses a dynamic number of LSH parameters (number of bands, number of shingles, and number of permutation lists) depending on the data similarity. The aim is to enhance matching accuracy to make it suitable for security applications such as forensics technology. The main contributions of this paper can be summarized as follows:

1. We performed a review of the recent advances in the LSH algorithm and summarized them based on the architectural framework, the applied application domain, and their contributions.
2. We proposed a new hashing algorithm (DLSH) based on the original LSH with an improvement in the matching accuracy. Dynamic identification of the original LSH parameters (number of bands, number of shingles, and number of permutation lists) is used in a hierarchical order to optimize searching results.

2. Related Works

There have been several improvements in the LSH algorithm. This section summarized a list of these improvements in the table below with the applications of LSH for each of the previous works.

Table 1: A list of previous works on the LSH improvements.

Related Works	LSH Technique	Distance measure	Application	Contributions
Kim, et al. [16]	Boosted LSH (BLSH)	Hamming distance	Audio spectra	Reducing the redundancy in LSH projections
Datar, et al. [7]	E2LSH	Euclidean distance	Synthetically generated data sets and query points	Reduces the number of false positives and false negatives while keeping the accuracy high.
Lv, et al. [17]	Hash-perturbation LSH	Euclidean distance	Image dataset and Audio dataset	Reduce the large number of required hash tables in the basic LSH
Lv, et al. [18]	Multi-probe LSH	Euclidean distance	Image dataset and Audio dataset	Uses less number of hash tables while achieving the same accuracy
Ji, et al. [19]	Super-bit LSH (SBLSH)	Angular distance	Image processing	Proposed a super bits method that results in a smaller estimation variance when the angle to estimate is within (0, π/2]
Bawa, et al. [20]	LSH Forest	Jaccard-based	Text/Document Processing	Creates a prefix tree on each hash table and stores the compound hash keys in the prefix trees
Satuluri and Parthasarathy [21]	BayesLSH	Euclidean distance, Angular, and Jaccard metrics	Text/Document Processing	Use Bayesian statistics to find the probability distribution of similarities between the query and candidates by knowing the distribution of collisions in projections.
Huang, et al. [22]	An optimised version of QALSH	Euclidean-based, Jaccard-based, and Hamming-based	Images processing	Uses compound hash keys and R-trees
Yu, et al. [23]	OS-LSH	Not mentioned	Audio	Improve the scalability of content-based retrieval of audio tracks in music databases.
Ozawa, et al.[24]	RAN-LSH	Tolerant distance	Security/Privacy	Detect the spam emails
Zhang, et al. [25]	LSHiForest	Any distance metric	Data Mining	The proposed method outperforms other LSH in time efficiency, anomaly detection quality, and robustness

Jiand and Sun [26]	Semi-Supervised SimHash	Hamming Distance	Text/Document Processing	Search for similar documents in high-dimensional spaces
Shrivastava and Li [27]	ALSH	Euclidean distance	Machine Learning	Improves the performance of Maximum Inner Product Search (MIPS)
Kim, et al. [28]	Stratified LSH (SLSH)	Various distance functions	Healthcare	Presents a detection system for high-dimensional physiological data using LSH
Fisichella, et al. [29]	SimPair LSH	Euclidean distance	Plagiarism/Near-Duplicate Detection	Solves the problem of near-duplicate detection for high dimensional data points incrementally and efficiently

By analyzing the related works from the literature, it can be said that the improvement in the LSH algorithm is done in the architectural framework of the original LSH algorithm to make it efficient for a specific application domain. As the LSH algorithm has been applied in many applications [30], the change in the base of the LSH algorithm has been done to improve its processing speed of it in various domains.

ln this paper, a new method is proposed which enhances the performance of the original LSH by adding value to the search technique aspect. The proposed algorithm is applied to the Reuters dataset to investigate its performance in searching for similar documents in a high-dimensional space. The new algorithm can be applied in many applications such as information retrieval and security applications that depend on searching high-dimensional data like forensics.

3. Methods and Materials

3.1. Locality Sensitive Hashing Algorithm

For a large dataset, D consists of n points and d dimensions, and for a query point q point to the same space of the dataset, the goal of Approximate Nearest Neighbors is to find an approximation ratio $c > 1$ such that for a returned point $o \in D$ the distance between two points is satisfied by the following formula:

$$dist(o, q) <= (c\ x\ o^*, q) \tag{1}$$

where o* is the true nearest neighbors of q in D [31]. The LSH algorithm uses a hash function to push down the nearest neighbors' points in a high-dimensional into low-dimensional space. For any two points x and y in the d-dimensional dataset, the hash function H is said to be sensitive if it satisfies the following conditions:

1) $|x - y| <= R, P(H(x) = H(y) >= p1$
2) $|x - y| <= cR, P(H(x) = H(y) <= p2$

where p1 and p2 are the probabilities and c is the approximation value. The conditions say that two points x and y that are close to each other will be hashed to the same bucket in the low-dimensional space with a very high probability >=p1. If they are not close, they will be hashed to the same bucket with a low probability <=p2. More details about how the LSH algorithm works are given below.

The original LSH algorithm hashes the data points in the text into buckets where the data points near each other are located on the same bucket, and the data points far from each other are in different buckets. By this, the degree of similarity between data points is increased. The algorithm consists of three main steps as follows [32, 33]:

1) Shingling: In this step, the input (for example, document) is divided into a set of characters of length k known as k-shingle to form the two-dimensional shingle matrix. A small value of k results in many shingles that may exist in the same document. Each document is represented as a column in the matrix and the set of shingles is represented in the rows. If the shingle exists in the document, a flag value is set to 1 in the matrix for the corresponding document.

After that, a similarity measure, the Jaccard index, is used to find the similarity between documents. A high Jaccard value means that the documents are most likely to be similar.

In finding similarities between documents, scalability is a major concern when having n large documents that will need a big memory for storing them and a high complexity to perform a comparison between shingles. To solve this issue, hashing is used to convert each document into a small signature value where the similarity between two documents implies a high probability of having similar hash values for the same documents. The hash function that links the similarity metrics used in LSH is the min-hash function which is applied in the next step.

2) Min-hashing: This step is critical and important. It is responsible for compressing the shingle matrix generated from the previous step into a smaller matrix called the signature matrix saving on the distance between the original documents.

The idea of the min-hashing algorithm depends on finding a number of random lists called permutation lists. Each permutation list refers to a hash function. Each permutation list produces a row in the signature matrix. The Min-hash function f (x) of any column is the number of the first row, in the permuted order, in which the column has a 1. The result of the Min-hashing process is stored in a signature matrix, where its rows are the Min-hashing value, and the column of the signature matrix is the file name.

The output of this step is the signature matrix with a compressed row. Each row represents a min-hash value and each column represents a document. The similarity between documents is the same as the original similarity between them in the shingle matrix.

3) Locality-sensitive hashing: This step divides the signature matrix into a set of bands in order to identify the similarity between documents. The details of this step are illustrated below:

a. Divide the signature matrix into b bands, each band having r rows

b. For each band, hash its portion of each column to a hash table with k buckets

c. Candidate column pairs are those that hash to the same bucket for at least 1 band

d. Tune b and r to catch most similar pairs but few non-similar pairs

3.2. The Proposed Algorithm: Dynamic Locality Sensitive Hashing Algorithm (DLSH)

The LSH algorithm is made up of a number of critical variables or factors that play a key role in the algorithm's accuracy and efficiency. The main important parameters of the LSH algorithm are:

1. The length of the shingle used in the shingling process.

2. The number of Bands.

3. The number of permutation lists used during the signature matrix creation process.

The proper tuning of these parameters influences the accuracy of the algorithm. In this paper, a new modification to the LSH algorithm has been proposed named Dynamic Locality Sensitive Hashing Algorithm (DLSH) that tunes these parameters dynamically in a hierarchical manner.

The DLSH algorithm is based on the old LSH algorithm, however, it has been improved in terms of efficiency and accuracy. The proposed technique does not rely on a fixed number of bands, shingles, or permutation lists. Instead, a dynamic form of these parameters is proposed to improve the accuracy of the original LSH algorithm.

These three parameters play an important role in the accuracy of the original LSH algorithm. The number of shingles divides the document into a specific number of shingles. Similar documents are more likely to share more shingles. In general, a small value of shingles will result in bad differentiation of documents and high time and space complexity.

Each shingle is then assigned to a unique index. The document can be represented as a binary vector with zero’s and one’s with one for every appearance of the shingle. For N number of documents and D total number of shingles, we have a huge matrix NxD.

After that, the permutation creates a signature for the document. To do so, several permutations are done on the document with different hash functions to generate the document signature. So, for N documents (row matrix) we perform K hash function (for each column) on the document. But, to make it more efficient, the algorithm actually does not permute all the rows. Instead, the algorithm performs band partitions.

The documents are hashed into buckets based on the hash function if it is 1 or 0. Having two documents in the same bucket means that these documents are more likely to be similar and can be considered as a candidate pair. Comparing the similarity between each pair of documents in N documents requires n2 operations. If 1 pair takes a microsecond, 5000 documents will take 10 seconds and 500,000 documents will require more than one day. So, the computational complexity of document comparison is a bottleneck of the algorithm.

The documents that appear as a candidate pair will be in the same band. The challenge here is how many rows in the signature matrix (number of a hash function) are to be used in the band, and how many bands can be used to divide the matrix. Tunning these parameters properly will affect the catching of similar pairs. In general, a higher number of bands implies lower similarity

In the proposed DLSH algorithm, a high accuracy rate is achieved by the proper tunning of the original LSH algorithm parameters. DLSH algorithm will test the accuracy over three parameters as follows:

The length of shingles will be defined as a range, for example, from 2 to 10. For each value in the range, the algorithm will run two inner loops to test the accuracy with a different number of bands and a different number of hash functions, which means that the number of bands will be started from two, then the algorithm will be executed over different hash functions, and the level of similarity is calculated for the specified number of shingle and bands. The best value is saved along with the optimal parameter value, then the number of shingles is increased and the algorithm repeats until the maximum loop parameter is reached. For any better value of accuracy achieved, the results are updated and the new value for the optimal parameter is modified.

As the DLSH algorithm is inspired by the original LSH algorithm, the shingling and min-hashing steps are the same for both of them. The last step is different, the details of the DLSH step are illustrated below:

- The process started by defining a range for shingles, which started as an example from 3 as the initial value for the number of shingles, and each time the number of shingles is incremented.

- For each file in the dataset or corpus, construct Vocab, which is a database that contains all shingles for all documents or files

- Specify the beginning value for Permutation lists, which, as previously noted, are responsible for converting the original term matrix into the small matrix, for the specified numbers of shingles. There will be a predetermined range for the number of permutation lists, which will be raised by a certain number, like three or two, each time.

- Construct the signature matrix.

- For each number of shingles and permutation lists, define a range for the number of bands started as an example from 3 and each time the number of bands is incremented. The bands are used to divide the signature matrix into groups, starting from Min number of bands, and hash each band into the bucket to define the matched files.

- The DLSH procedure will start with an initial value of Permutation lists that will be used to calculate the similarity between the target file or tampered file and all other files and specify the files that achieve a high level of accuracy based on the equation (1). The parameter values that were utilized to reach this degree of accuracy using current parameters will be stored, such as the number of shingles, bands, permutation lists, accuracy value, and similarity findings, will be stored with the accuracy of the matching process with the tampered file.

- The similarity ratio between tampered file and all other files will be calculated with each number of permutation lists until reaches the maximum number of permutation lists.

- The number of bands will be increased by a specified range, and start again with an initial number of Permutation Lists, and the accuracy will be calculated and stored with used parameter values.

- The number of shingles will be increased by a specified range with the number of bands reached to the maximum number of bands, new accuracy values with each number of shingles, bands, and permutation lists.

- Until the number of shingles reaches the maximum values or the accuracy with target files reaches the optimal values that are established as a threshold, the algorithm will continue to calculate accuracy and similarity.

- At the end, the algorithm will generate the percentage of matching between tampered and target files, the accuracy level for the files which participated in tampered file content, and the optimal value of parameters used to achieve the optimal accuracy value or the maximum value for accuracy based on a comparison between different achieved results.

Figure 1 below represents these steps as a flowchart and shows the relations between different steps.

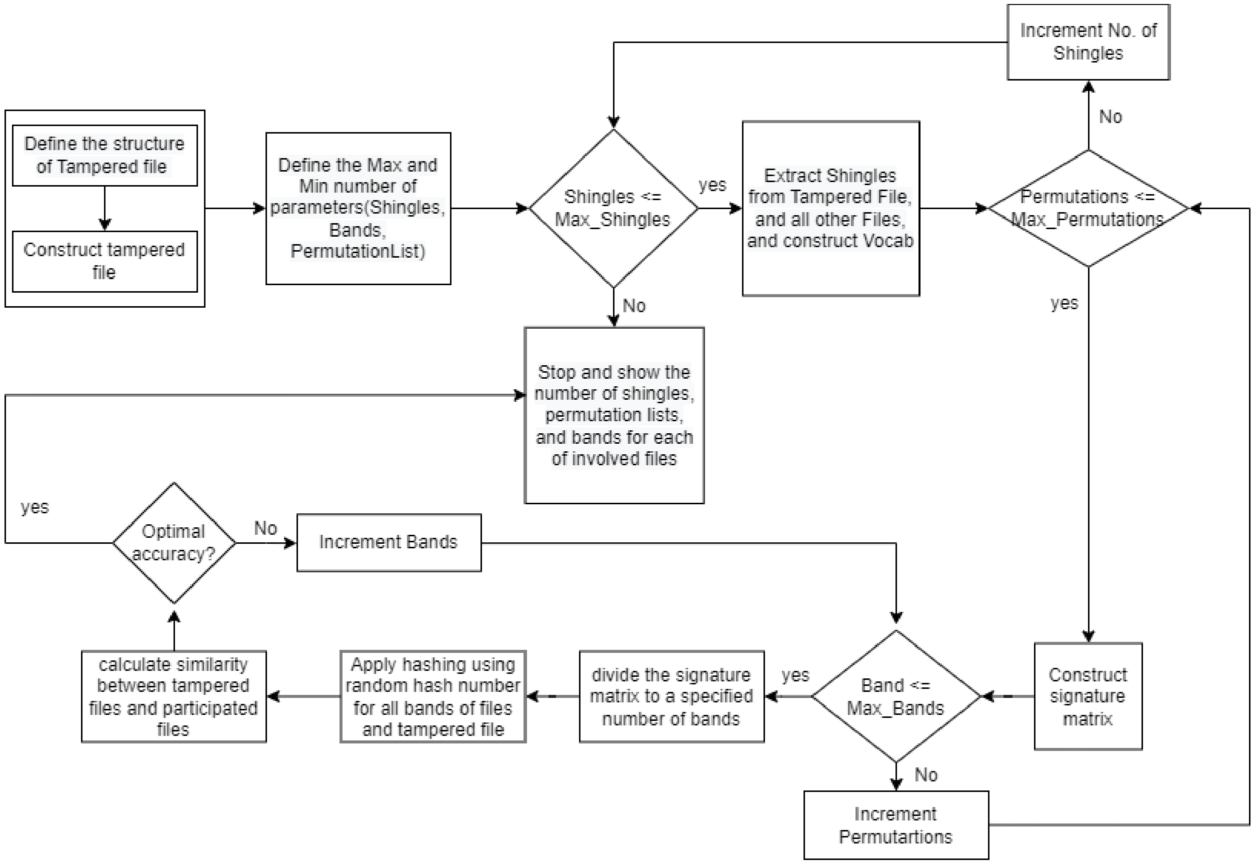
Define the structure of Tampered file
Construct tampered file
Define the Max and Min number of parameters(Shingles, Bands, PermutationList)
Shingles <= Max_Shingles
yes
No
Extract Shingles from Tampered File, and all other Files, and construct Vocab
Permutations <= Max_Permutations
No
Increment No. of Shingles
yes
Stop and show the number of shingles, permutation lists, and bands for each of involved files
yes
Optimal accuracy?
No
Increment Bands
Construct signature matrix
Band <= Max_Bands
yes
No
Increment Permutartions
divide the signature matrix to a specified number of bands
Apply hashing using random hash number for all bands of files and tampered file
calculate similarity between tampered files and participated files

Figure 1: Main steps of dynamic locality sensitive hashing (DLSH).

3.3 Material Dataset

The idea of the proposed DLSH algorithm has been tested and evaluated on the Reuters dataset. The Reuters dataset is a collection of documents with news articles. The original corpus has 10,369 documents and a vocabulary of 29,930 words. The Reuters dataset is used to construct a set of tampered files that are used for the testing of algorithm performance. To investigate the performance of the proposed DLSH searching algorithm, we construct a tampered file from the dataset and let the algorithm search the whole database for it. The tampered file is constructed randomly, which means that it can be defined as a collection of pieces of data from different files from the original dataset where each file participates in the tampered file construction with a specific percentage. The data shouldn't be in sequence and is selected from the original file in the dataset randomly.

3.3.1 Tampered File Structure

The methodology considered in this experiment is very efficient and can be used in many critical applications such as forensics technology where finding any piece of information is very important to determine if this information has been altered or not. Evidence tampering refers to the situation where the attacker can falsify and alter the evidence to interfere with the investigation process of forensics, which is known as spoliation. The evidence tampering affects the final verdict. The important issue here is that the investigator requires a powerful technique to ensure that tempering happens even if it is minor or not. It is known that comparing digital evidence in a huge dataset to determine and identify tampering requires a lot of resources with additional security requirements.

For this reason, we investigate the performance of the proposed algorithm DLSH over a larger dataset and by using a different tampered file structure. The aim is to create a tampered file by adding a different piece of data from different locations and test the performance of DLSH in searching the whole dataset to find the matched data.

Six file structures are used to construct the tampered file before running the experiments. The aim of using this method is to show the effectiveness of the proposed algorithm in finding the match of each piece of file with its corresponding in the big dataset. In the first case, the tampered file is constructed from two different files with different participation percentages for each one. The participation of file manipulation in the second structure consists of three files. The third structure consists of four files. The fourth and fifth structures consist of six and eight files participation respectively. Finally, the sixth structure consists of ten files in the tampered file structure.

4. Experimental Results and Discussion

In this section, the performance of the proposed DLSH algorithm is given by applying it to the tampered files and Reuter dataset for searching for the original file. Six main experiments were done using the six different tampered file structures. Each time the algorithm searches for the best tunning of the length of the shingle, the optimal number of bands, and the optimal number of permutations. The best accuracy value is saved with the value of the optimal parameter.

The accuracy is evaluated as follows:

$$abs(100 - \frac{abs|Similarity\ Ratio\ between\ two\ files*100-OptimalSimilarityRatio|}{OptimalSimialrityRatio} * 100 \qquad (2)$$

The results are given in the tables below.

Table 2: The results generated by the DLSH algorithm in searching for the tampered file with two different file’s structure

File number	Participation percentage	Accuracy Rate	Similarity Rate	Optimal number of bands	Optimal number of Shingles	Optimal number of permutations
File One	62%	98.56%	61.11%	18	8	18
File Two	38%	97.70%	38.88%	18	9	18

Table 3: The results generated by the DLSH algorithm in searching for the tampered file with three different file’s structure

File number	Participation percentage	Accuracy Rate	Similarity Rate	Optimal number of bands	Optimal number of Shingles	Optimal number of permutations
File One	35%	95.23%	33.34%	3	3	3
File Two	41%	98.39%	41.66%	12	5	12
File Three	24%	96.00%	25.00%	12	4	12

Table 4: The results generated by the DLSH algorithm in searching for the tampered file with four different file’s structure

File number	Participation percentage	Accuracy Rate	Similarity Rate	Optimal number of bands	Optimal number of Shingles	Optimal number of permutations
File One	21%	96.61%	22.22%	9	3	7
File Two	27%	98.76%	26.67%	15	10	15
File Three	20%	99.20%	20.83%	24	8	24
File Four	30%	99.42%	29.17%	24	6	24

Table 5: The results generated by the DLSH algorithm in searching for the tampered file with six different file’s structure

File number	Participation percentage	Accuracy Rate	Similarity Rate	Optimal number of bands	Optimal number of Shingles	Optimal number of permutations
File One	15%	98.77%	14.814%	27	19	27
File Two	25%	100.00%	25.000%	12	5	24
File Three	12%	96.00%	12.500%	24	19	24
File Four	28%	99.20%	27.777%	18	10	18
File Five	13%	97.50%	13.334%	15	14	15
File Six	7%	95.23%	6.667%	15	3	15

Table 6: The results generated by the DLSH algorithm in searching for the tampered file with eight different file’s structure

File number	Participation percentage	Accuracy Rate	Similarity Rate	Optimal number of bands	Optimal number of Shingles	Optimal number of permutations
File One	12%	96.00%	12.50%	24	6	24

File Two	14%	95.23%	13.33%	15	5	15
File Three	22%	99.00%	22.22%	9	3	18
File Four	11%	99.00%	11.11%	9	4	18
File Five	27%	98.77%	26.67%	15	14	15
File Six	5%	95.23%	4.76%	21	8	21
File Seven	6%	90.00%	6.67%	15	4	15
File Eight	3%	72.00%	4.17%	24	18	24

Table 7: The results generated by the DLSH algorithm in searching for the tampered file with ten different file's structure

File number	Participation percentage	Accuracy Rate	Similarity Rate	Optimal number of bands	Optimal number of Shingles	Optimal number of permutations
File One	8%	96.00%	8.333%	12	17	24
File Two	12%	96.00%	12.500%	24	19	24
File Three	6%	92.95%	5.555%	18	14	18
File Four	14%	84.00%	16.666%	6	3	12
File Five	7%	94.50%	7.407%	27	11	27
File Six	13%	97.50%	13.333%	15	7	15
File Seven	5%	60.00%	8.334%	12	5	24
File Eight	15%	98.77%	14.814%	27	9	27
File Nine	4%	72.00%	5.555%	18	21	18
File Ten	16%	96.00%	16.667%	6	3	18

Analyzing the results that are presented in the tables shows that the DLSH algorithm is accurate in the process of matching all the files that were involved in the tampered file construction process. The level of accuracy in the matching with original files was calculated using the equation mentioned earlier.

However, because more than one structure of tampered files is used to test the ability of the algorithm to extract the original files, the matching procedure is difficult. The results reveal that in most circumstances, the accuracy of matching in each case of a tampered file is greater than 80% to identify the original files from thousands of files. The file size influences the accuracy rate, the file that has higher participation in the tampered file returns a higher accuracy rate and vice versa. In general, a file with a small percentage of participation needs a high band value to generate the most accurate rate. This result is important because if we have a large number of bands; this means that we increase the probability of finding a pair of documents that are likely to be similar. As this happens with a small piece of the document (low participation percentage) this means that the algorithm is able to search on this small piece accurately and find the matching documents. Thus, DLSH algorithm is robust and more efficient than regular LSH.

Another observation is related to the number of permutations being close to the number of bands. The proper tunning of these parameters is powerful. It is well known in the LSH algorithm that if we take a large value of bands means that a greater number of permutations (hash functions) will be used. DLSH finds this result heuristically with its robust search. The value of the number of bands found by the algorithm is close to the number of permutations in each experiment.

DLSH algorithm is robust, optimized, and able to tune its parameters heuristically to guarantee the generation of highly accurate matching results. This conclusion makes the algorithm suitable to be applied in many important applications where searching for a huge amount of data is mandatory. Forensic technology is an example of such an application. The investigator in forensics technology may need to search a huge amount of data for a small piece of information to be used as evidence in the criminal investigation.

5. Conclusion and Future Works

This paper proposed a new Hierarchical Locality Sensitive Hashing algorithm (DLSH), which is based on the original Locality Sensitive Hashing algorithm (LSH). The DLSH algorithm tunes some parameters of the original LSH algorithm heuristically by searching for the optimal value of these parameters that will lead to the best searching accuracy score. These parameters are the length of the shingle, the number of bands, and the number of permutation lists. The new algorithm was tested using different structures for tampered file content. The performance evaluation was done by searching for the matching between created tampered files over the Reuters dataset which contains 10,369 documents and a vocabulary of 29,930 words. The results show that DLSH algorithm is robust and accurate in the matching process with a high score for accuracy value.

In the future, the DLSH algorithm could be employed in a variety of sectors that rely heavily on the concept of matching, such as digital forensics, intrusion detection systems, and other sensitive fields. Other improvements to this version of the DLSH algorithm may be added in the future to improve accuracy and minimize execution time.

References

[1] Bello-Orgaz, G.; Jung, J.J.; Camacho, D. Social big data: Recent achievements and new challenges. Inf. Fusion 2016, 28, 45–59

[2] Jinfeng Li, James Cheng, Fan Yang, Yuzhen Huang, Yunjian Zhao, Xiao Yan, and Ruihao Zhao. 2017. Losha: A general framework for scalable locality sensitive hashing. In Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. 635–644.

[3] G Padmasundari and Hema A Murthy. 2017. Raga identification using locality sensitive hashing. In 2017 Twenty-third National Conference on Communications (NCC). IEEE, 1–6

[4] Brian D Ondov, Todd J Treangen, Páll Melsted, Adam B Mallonee, Nicholas H Bergman, Sergey Koren, and Adam M Phillippy. 2016. Mash: fast genome and metagenome distance estimation using MinHash. Genome biology 17, 1 (2016), 1–14.

[5] [5] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. 2001. Searching in metric spaces. ACM computing surveys (CSUR) 33, 3 (2001), 273–321

[6] Park, J.S.; Chen, M.S.; Yu, P.S. An Effective Hash-Based Algorithm for Mining Association Rules; ACM: New York, NY, USA, 1995

[7] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In Proceedings of the twentieth annual symposium on Computational geometry.

[8] Mehmet Ali Abdulhayoglu and Bart Thijs. 2018. Use of locality sensitive hashing (LSH) algorithm to match Web of Science and Scopus. Scientometrics 116, 2 (2018), 1229–1245

[9] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH forest: self-tuning indexes for similarity search. In Proceedings of the 14th international conference on World Wide Web. 651–660.

[10] Gan, Junhao, et al. "Locality-sensitive hashing scheme based on dynamic collision counting." Proceedings of the 2012 ACM SIGMOD international conference on management of data. 2012.

[11] Jafari, Omid, et al. "A survey on locality sensitive hashing algorithms and their applications." arXiv preprint arXiv:2102.08942 (2021).

[12] Wanqi Liu, Hanchen Wang, Ying Zhang, Wei Wang, and Lu Qin. 2019. I-LSH: I/O efficient c-approximate nearest neighbour search in high dimensional space. In 2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 1670–1673.

[13] Sunwoo Kim, Haici Yang, and Minje Kim. 2020. Boosted Locality Sensitive Hashing: Discriminative Binary Codes for Source Separation. In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 106–110.

[14] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In Proceedings of the 2012 ACM SIGMOD international conference on management of data. 541–552

[15] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In Proceedings of the thirtieth annual ACM symposium on Theory of computing. 604–613.

[16] Sunwoo Kim, Haici Yang, and Minje Kim. 2020. Boosted Locality Sensitive Hashing: Discriminative Binary Codes for Source Separation. In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 106–110.

[17] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. [n.d.]. A Time-Space Efficient Locality Sensitive Hashing Method for Similarity Search in High Dimensions. Technical Report.

[18] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2007. Multi-probe LSH: efficient indexing for high-dimensional similarity search. In 33rd International Conference on Very Large Data Bases, VLDB 2007. Association for Computing Machinery, Inc, 950–961.

[19] Jianqiu Ji, Jianmin Li, Shuicheng Yan, Bo Zhang, and Qi Tian. 2012. Super-bit locality-sensitive hashing. In Advances in neural information processing systems. Citeseer, 108–116.

[20] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. 2005. LSH forest: self-tuning indexes for similarity search. In Proceedings of the 14th international conference on World Wide Web. 651–660.

[21] Venu Satuluri and Srinivasan Parthasarathy. 2011. Bayesian locality sensitive hashing for fast similarity search. arXiv preprint arXiv:1110.1328 (2011).

[22] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. Proceedings of the VLDB Endowment 9, 1 (2015), 1–12.

[23] Yi Yu, Michel Crucianu, Vincent Oria, and Ernesto Damiani. 2010. Combining multi-probe histogram and order-statistics based lsh for scalable audio content retrieval. In Proceedings of the 18th ACM international conference on Multimedia. 381–390.

[24] Seiichi Ozawa, Junji Nakazato, Tao Ban, Jumpei Shimamura, et al. 2015. An online malicious spam email detection system using resource allocating network with locality sensitive hashing. Journal of intelligent learning systems and applications 7, 02 (2015), 42.

[25] Xuyun Zhang, Wanchun Dou, Qiang He, Rui Zhou,

Christopher Leckie, Ramamohanarao Kotagiri, and Zoran Salcic. 2017. LSHiForest: A generic framework for fast tree isolation based ensemble anomaly analysis. In 2017 IEEE 33rd International Conference on Data Engineering (ICDE). IEEE, 983–994

[26] Qixia Jiang and Maosong Sun. 2011. Semi-supervised simhash for efficient document similarity search. In Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies. 93–101.

[27] Anshumali Shrivastava and Ping Li. 2014. Asymmetric LSH (ALSH) for sublinear time Maximum Inner Product Search (MIPS). Advances in Neural Information Processing Systems 3, January (2014), 2321–2329.

[28] Yongwook Bryce Kim, Erik Hemberg, and Una-May O'Reilly. 2016. Stratified locality-sensitive hashing for accelerated physiological time series retrieval. In 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). IEEE, 2479–2483.

[29] Marco Fisichella, Fan Deng, and Wolfgang Nejdl. 2010. Efficient incremental near duplicate detection based on locality sensitive hashing. In the International Conference on Database and Expert Systems Applications. Springer, 152–166

[30] Jafari, Omid, et al. "A Survey on Locality Sensitive Hashing Algorithms and their Applications." arXiv preprint arXiv:2102.08942 (2021).

[31] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In Proceedings of the thirtieth annual ACM symposium on Theory of computing. 604–613

[32] D. Ravichandran, P. Pantel, and E. Hovy. Randomized algorithms and nlp: using locality sensitive hash function for high speed noun clustering. In ACL, 2005.

[33] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In Proc. of 25th Intl. Conf. on Very Large Data Bases(VLDB), pages 518–529, 1999