Similarity Search using Locality Sensitive Hashing and Bloom Filter

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Computer Science and Engineering

Submitted By
Sachendra Singh Chauhan
(Roll No. 801232021)

Under the supervision of
Dr. Shalini Batra
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2014

CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, **"Similarity Search using Locality Sensitive Hashing and Bloom Filter"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of ***Dr. Shalini Batra*** and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature: Sachendra

(Sachendra Singh Chauhan)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(**Dr. Shalini Batra**)
Assistant Professor,
Computer Science and Engineering Department

Countersigned by

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

ACKNOWLEDGEMENT

No volume of words is enough to express my gratitude towards my guide, ***Dr. Shalini Batra,*** *Assistant Professor, Computer Science and Engineering Department, Thapar University, who has been very concerned and has aided for all the material essential for the preparation of this thesis report. She has helped me to explore this vast topic in an organized manner and provided me with all the ideas on how to work towards a research-oriented venture.*

I am also thankful to ***Dr. Deepak Garg****, Head of Department, CSED and* ***Dr. Ashutosh Mishra****, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.*

I would also like to thank the faculty members who were always there in the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis.

Most importantly, I would like to thank my parents and the Almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Sachendra Singh Chauhan
801232021

Abstract

Similarity search of text documents can be reduced to Approximate Nearest Neighbor Search by converting text documents into sets by using Shingling. In the proposed scheme conversion of a document into set is done by using k-shingle and similarity between sets, is calculated using Jaccard similarity. Characteristic matrix, created by searching shingles in each document is generated using the shingles and hash functions. Since the complexity of the matrix is significantly high when shingle set is very large, a technique called 'minhashing' is used to reduce the size of the matrix. Search time can be drastically reduced if characteristics matrix are created by using Bloom Filter. Bloom Filter is a probabilistic data structure that will reduce O(n) search time to constant time. Minhashing creates a signature matrix which is very less in size as compared with characteristics matrix but gives almost same result. Further, finding the similarity among all pairs of column of signature matrix is still a big problem because comparing n columns take $O(n^2)$ time. The time of comparing columns for similarity is reduced by using another technique, Locality Sensitive Hashing. The proposed scheme has been successfully implemented and comparative analysis of existing methodology and proposed methodology has been provided.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Approximate Nearest Neighbor
APSS	All Pair Similarity Search
BoW	Bag of Words
BM	Bloom Filter
CM	Characteristic Matrix
C2LSH	Collision Counting Locality Sensitive Hashing
D	Document
DM	Distance Measure
DS	Document Set
J-DIS	Jaccard Distance
J-SIM	Jaccard Similarity
LCS	Longest Common Subsequence
LSH	Locality Sensitive Hashing
NNS	Nearest Neighbor Search
PLEB	Point Location in Equal Balls
S	Shingle
SM	Signature Matrix
SN	Shingle Number

Chapter 1
Introduction

The limitless heterogeneity of internet resources and the recent advances in the areas of computation and communications have forced the overloading of digital libraries and data warehouses. Extracting useful information from these datasets is very time consuming. The basic operation required to retrieve desired information is searching. One of the most basic form of search is string search where speed is the most important feature of a given algorithm and indexing is usually not worth the effort to text retrieval in huge document collections. But in today's world as numbers of applications are increasing, there is need to search more complex data objects like images, videos, graphs, networks, documents, web pages etc. Depending on the application, this either makes the search problem more complex since it needs to do more work to adapt to the type of data or it makes it easier because we can exploit additional information.

1.1 Similarity Search

The Metric Space Approach focuses on finding the efficient ways for locating user-relevant information in collections of objects, the similarity of which is quantified using a pair wise distance measure.

Similarity search has become a primary computational task in a range of application areas, including pattern recognition, data mining, biomedical databases, multimedia information retrieval, machine learning, data compression, computer vision and statistical data analysis. The exact match rarely has meaning in these environments whereas proximity/distance concepts (similarity/dissimilarity) are typically much more beneficial for searching.

1.1.1 Vector Spaces

Several similarity search applications represent data as high dimensional vectors. Typically, similarity between any two texts or images is assessed by comparing the features extracted by them. For example, colour histogram, a typical image feature can be

represented as a vector, and where the density of the colour associated with dimension [1] is the value contained in each dimension. When the data is represented in vector spaces, the similarity between data can be calculated by measuring the distances between the two vectors using various functions such as Euclidean distance, Jaccard distance, *etc*.

1.1.2 Metric Spaces

Some similarity search applications have two basic problems- vector representation of data is not effective and Minkowski family functions fail to effectively define the distance measure. For example, consider the colour histogram similarity search problem. Minkowski functions compare values independently of the others in each dimension. However, colour histogram similarity can be better assessed with the quadratic form distance [2,3] which in fact, does not belong to the Minkowski family. Therefore, data can be represented in metric space in such cases. The metric space has an advantage that no specific prerequisites are desired for representation of data but the distance function only need to follow the symmetric, positivity, identity, and triangular inequality properties.

1.2 Lexical Similarity

In linguistics, lexical similarity can be defined as the degree of similarity among the two given word sets of any two given languages. The lexical similarity of 1 (or 100%) means the overall similarity among vocabularies and lexical similarity of 0 means there are no common words.

1.2.1 Shingling

Shingling [4] refers to the detection of near-duplicate items. Given a positive integer k and a sequence of terms in a document D, the k-shingles of D are defined as a set of all the successive sequences of k terms in D. The general approach followed to convert a document into a set is to shingle the document. k-shingles define a set of all k size non repeatable substrings of the document, and group them is a single object. The set of k-shingles of a document with n words takes space O(kn). The space goes on decrease as items are repeated in the document. For example, if k=3 and a document contains text

"*This LSH Project is good*" then the shingle's set will be {"This LSH Project", "LSH Project is", "Project is good"}.

1.2.2 Bag of Words

The methodology of Bag-of-words (BoW) was first adduced for text document analysis for the text retrieval domain problem. BoW model uses a visual analogue of a word for image analysis, based on the vector quantization process where the low-level visual features of local points, such as colour, texture etc are clustered. It involves the under mentioned four steps for extracting the BoW feature [3] from the images:

- Detecting points of interest automatically,
- Computing local descriptors over those points,
- The descriptors are quantized into words to form the visual vocabulary
- The occurrence of each specific word of vocabulary in the image is found which can be used to construct the BoW feature.

Figure 1.1 describes sequence of steps for extracting BoW feature from images.

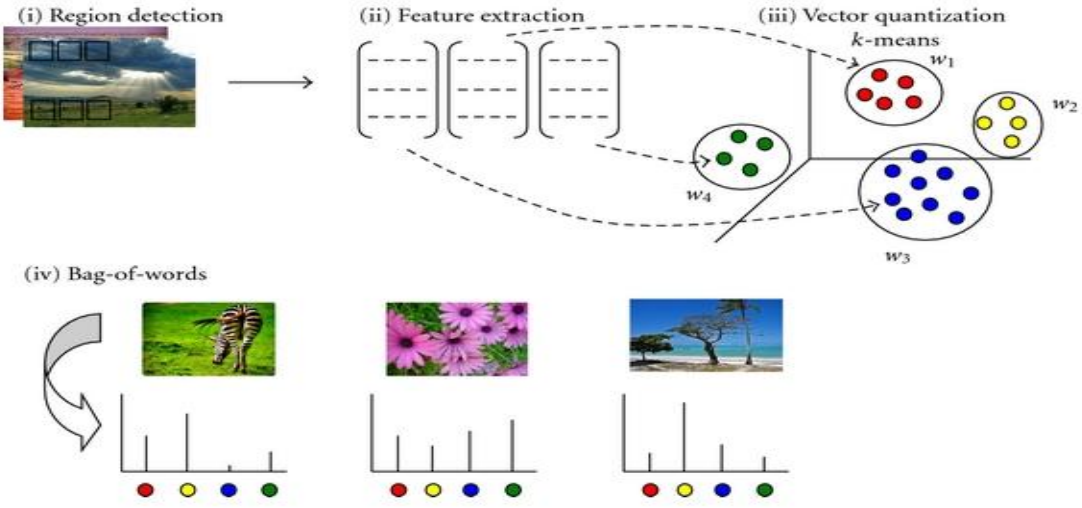

Figure 1.1: Steps to construct the BoW for image [3]

1.3 Semantic Similarity

Semantic similarity is defined over a set of text files where the distance among files is achieved from the equality of their semantic content as compared to similarity based on their syntactical representation (i.e. string type). The semantic relationship strength

among units of language can be estimated by the use of these mathematical tools. This can be done with the help of numerical description which is obtained by comparing the information formally or implicitly supporting their meaning. Specifically, it is likely to assess semantic similarity by describing a topological similarity, which uses various ontologies for estimating the distance between concepts or terms. For example, the concepts are characterized as the nodes of a directed acyclic graph in a semi-ordered set. The metric which can be used to compare the concepts in this case is the shortest-path which links the two concept nodes. Semantic relatedness between words or group of words can be predicted with the help of various methods such as a vector space model.

1.4 Duplicate

There are abundant of duplicate web pages in the mirror sites on the internet. Two such documents may differ from each other only by small portion of text. So it is very important to find and remove such duplicates search results to enhance the reliability of internet sources.

1.4.1 Exact Duplicate

It is very easy to test for the two documents if they are exactly similar or not. The two documents are compared character by character to test for exact duplicates and they are not same if they ever differ. However it is not the right method to test the similarity of the documents. It is very convenient for removing the redundancy in the system. This technique is used to find and remove duplicate documents, audio, photos, etc. Figure 1.2 shows the exact duplicates of documents in the selected folder. The duplicates of the documents are checked and can be deleted from the folder. The deleted duplicates files are shown in figure 1.3.

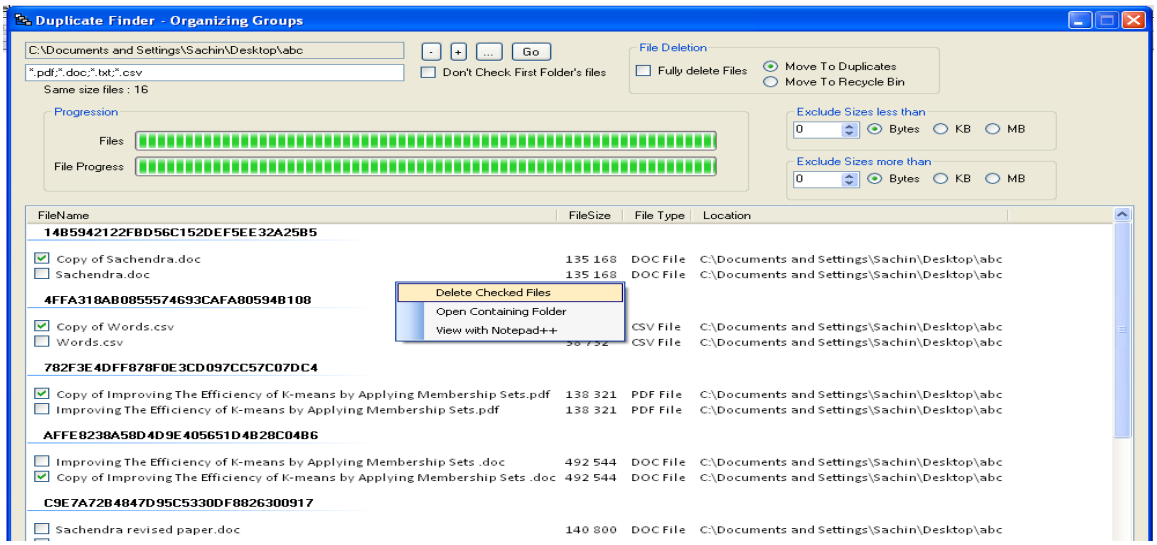

Figure 1.2: Testing the exact duplicate for documents

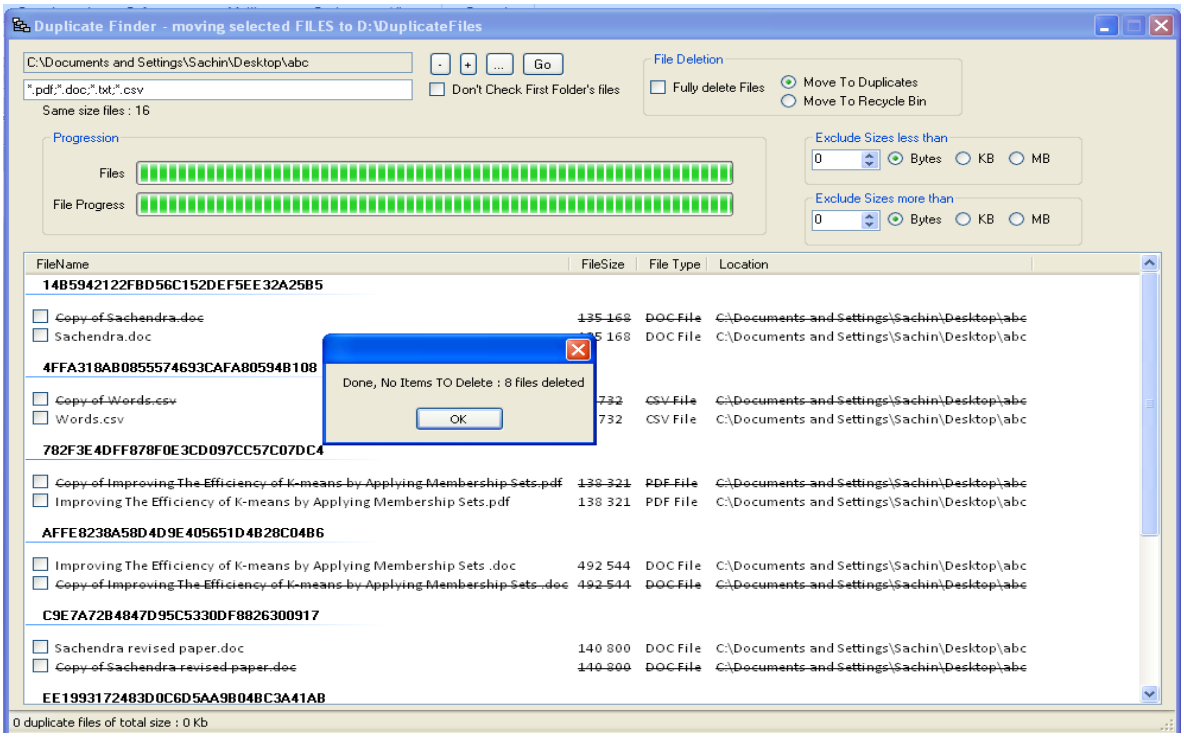

Figure 1.3: Deleting the exact duplicates of the documents.

1.4.2Near Duplicate

Near-duplicates are based on the identical primitive text which is altered and post processed and results in two discrete files. Near-duplicates can also be related to showing

of the identical scene or event. Multimedia content analysis faces objection for the detection of near-duplicates. Both informational and redundant signals are carried by near-duplicate. For example, near duplicates may supply rich visual evidences for cataloguing and summarizing videos from various origins. On the contrary, browsing of streaming web videos over internet is an intense time consuming task due to the exaggerated amount of near-duplicates. Subsequently, academia, industry and government organizations have shown a keen interest in near-duplicates for use in numerous multimedia applications and for the web scale search, detection and elimination of near-duplicates.

1.4.3 k-most near similar items to query

Given a query object and similarity described as distances in a metric space, the task of computing the K most similar objects is commonly termed as the k-nearest-neighbor (k-NN) procedure [5]. There are varieties of classification and regression tasks on which this procedure can be applied. Despite, its implementation experiences the high computation overheads, which requires N distance evaluations for each test data point. To accelerate this procedure, various space partitioning methods such as KD-Trees [6] and M-Trees [7] have been propounded, providing the fast exact K-nearest neighbor retrieval [8, 9].

1.4.4 All pair similarity search problem

The all pairs similarity search problem [10] can be defined as the problem to search all the pairs of items which have the similarity value above the predefined value. Today, many applications in data sciences like digital libraries, recommender systems and search engines need to solve this problem of datasets as these applications have ample of entries in a high dimensional space, that are usually scarce.

1.5 Distance Measures

A distance measure [11] is a function $dm(p_1, p_2)$ which takes the two arguments, some points in space as input and returns a real number, satisfying the following conditions:

- $dm(p_1, p_2) \geq 0$
- $dm(p_1, p_2) = 0$ only if $p_1 = p_2$
- $dm(p_1, p_2) = dm(p_2, p_1)$

- $dm(p_1, p_2) \leq dm(p_1, p_3) + dm(p_3, p_2)$.

There are different Distance Measures used to calculate the closeness of the points in some given space. Some of the popularly used distance measures are given below;

1.5.1 Jaccard Distance

The Jaccard Distance J-DIS(S_1,S_2), between two sets is defined as one minus Jaccard similarity between those two sets i.e. J-DIS (S_1, S_2) = 1 – J-SIM(S_1, S_2). Jaccard distance is defined for sets. Jaccard Similarity J-SIM(S_1, S_2) between two sets is calculated as the ratio between the intersection size and the union size of the sets.

Jaccard Distance has all the constraints of a distance measure-

- Since the size of intersection is always less than or equal to the size of the union, Jaccard Distance satisfies the non-negative constraints.
- The size of union and intersection of two sets can never be same at the same time except the case when both sets are same. Jaccard Similarity is one only when same sets are used. Further, Jaccard Distance is strictly positive.
- Union and intersection of two sets are always symmetric; Jaccard Distance satisfies symmetric axiom.
- Jaccard Similarity always satisfies triangular inequality, and so does Jaccard Distance.

1.5.2 Euclidean Distances

Let there is an n-dimensional Euclidean space S. Each point in this space is represented as a vector of n real numbers. Then the Euclidean distance between any two points x and y, d(x,y), is defined in equation (1)

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \qquad (1)$$

This distance is called the L_2 norm.

There are other variants of this distance, defined as L_R-norm. L_R-norm for Euclidean space is defined as:

$$d([x_1, x_2, \ldots, x_n],\ [y_1, y_2, \ldots, y_n]) = (\sum_{i=1}^{n} |x_i - y_i|^r)^{1/r} \qquad (2)$$

A common distance measure is Manhattan distance called the L1-norm where the sum of magnitudes of the difference in each dimension is the distance between two points.
It satisfies all the four previously defined axioms of a distance measure function-

- Since square root of a real number can never be negative. So non-negative constraint is satisfied for Euclidean Distance.
- The square of difference between any two real numbers cannot be 0, except the case when both numbers are same. So the positive distance constraint for Euclidean Distance is satisfied.
- Since $(x_i - y_i)^2 = (y_i - x_i)^2$, so Symmetry for Euclidean Distance is satisfied.
- The sum of the lengths of any two sides of a triangle is always greater than or equal to the length of the third side. So Euclidean Distance satisfies triangular inequality.

1.5.3 Cosine Distance

The cosine distance [11] between two points is the angle formed between their vectors. The angle always lies in between 0 to 180 degree regardless of the number of dimensions. The Cosine distance is measurable in Euclidean spaces of any dimensional spaces where vectors of points in spaces are integer or Boolean components. In this case, points can be considered as directions. The cosine of the angle between two vectors is calculated as the dot product of vectors by dividing with the L2-norms of x and y. Cosine distance is a distance measure-

- Since the values are taken in the range of (0,180), so there cannot be any negative distances.
- Angle between two vectors is 0 only if they are in the same direction.
- Angle between two vectors satisfies symmetry.
- Cosine distance also satisfies triangle inequality.

1.5.4 Hamming Distance

The Hamming Distance is used for the Boolean vectors i.e. which contain only 0 or 1. The number of items in which the two items differ is the hamming distance between them. Hamming distance is definitely a distance measure.

- The Hamming distance can never be negative.
- Hamming distance is zero, only if the vectors are identical.
- The hamming distance is same irrespective of order of the vectors and does not depend on which of two vectors we take first. So it is symmetrical.
- The triangle inequality is also satisfied because if x is the number of component in which p and r differ, and y is the number of component in which r and q differ, then p and q cannot differ in more than x + y components.

1.5.5 Edit Distance

Edit distance is used for the string points in vector space. Considering the two strings $x = x_1 x_{2....} x_n$ and $y = y_1 y_{2....} y_m$, the minimum number of times the insertions and deletions of single characters is done to convert x to y or y to x is the distance between them. The longest common subsequence (LCS) of x and y can also be used to calculate edit distance. Which can be obtained by subtracting twice the length of their LCS from the sum of lengths of x and y. Edit distance is a distance measure because:

- To convert string x into string y, number of insertions and deletions can never be negative.
- Edit Distance between two strings is zero only if the two strings are same.
- Number of insertions and deletions when converting string x to y or string y to x is same.
- Number of insertions and deletions when converting string x to z plus z to y will always be greater than or equal to that of converting string x directly to y.

1.6 Structure of the Thesis

The rest of the thesis is organized in the following order:

Chapter 2 gives the literature reviewed and chapter 3 defines the problem statement for this work. Chapter 4 introduces the LSH and Bloom Filter. The design techniques are

given in chapter 5 and implementation and results in chapter 6. Finally, chapter 7 gives the conclusion, summary of contribution and future scope of the work.

Chapter 2
Literature Review

The nearest neighbor search algorithm performs well for less-dimensional data, but the complexity in both criteria (time and space) increases exponentially as the dimensions goes high, known as "curse of dimensionality". All existing available solutions provide little improvement over linear search that compares the asked query to each entry in the dataset.

In [12] it is shown that for sufficiently high dimensions all available partitioning schemes downgrades to linear search. There are many tree-based indexing schemes for Nearest Neighbor search, K-D trees [6], navigating nets [13], cover trees [14], R-trees [15], and SR-trees [16], *etc*. But, these indexing schemes do not perform well when dimension of data increases, and it has been proved in [12] that for high dimensions these methods show only little improvement over linear search.

In [12], Indyk and Motwani proposed a hashing based high-dimensional approximate similarity search scheme called Locality-Sensitive Hashing (LSH). This scheme is linearly depends on the size of the dataset. Instead of partitioning method, it uses several hashing methods to hash the points to increase the probability of collision for objects which are similar than for those which are not. Then near neighbors can be determined by hashing the queried point and retrieving those elements stored in buckets that contain those points. This LSH scheme is applied when the points are given in binary Hamming space. The fundamental drawback of LSH is that it is fast and simple only when the points are available in the Hamming space. It is further discussed that the algorithm can be used so that data can be extended to be in hamming distance, but it is achieved at time complexity cost and in increased error (by a large factor).

In [17] an improvement in the running time of the earlier algorithm has been purposed for the L2 norm which results in an efficient Approximate Nearest Neighbor scheme (ANN) for the case L_p norm ($p<1$). It takes O(log(n)) to search exact near neighbor for data having growth boundary conditions. The proposed LSH algorithm works directly on data points distributed in the Euclidean space and do not perform any per-processing. The proposed scheme gives 40 times faster results than kd-tree.

In [18] Gan *et al.* proposed an LSH algorithm, Collision Counting LSH (C2LSH) that uses a m-base LSH functions to form dynamic compound hash functions instead of traditionally used static compound hash function. In C2LSH a collision threshold is used for data object to increase the query quality.

Broder *et al.* [19] designed an LSH scheme based on Jaccard similarity of sets which uses Minhash functions to take different permutations to calculate Jaccard similarity for sets [10]. Several variations have been proposed to improve the performance of this LSH scheme in [19].

In [21], Panigrahy *et al.* proposed an LSH scheme which is entropy-based. It minimizes the number of hash tables required, by looking up a number of query offsets in addition to the query itself. In addition to considering the bucket corresponding to the query, buckets corresponding to perturbed versions of the query also considered. Unfortunately, while the space requirements are reduced, the query time is considerably increased.

In [22] Dasgupta *et al.* proposed a technique to improve the calculation of Euclidean distance of d-dimensional data by using randomized Hadamard transforms. It gives the time complexity of (k, L)-parameterized LSH O(d log d + kL) which in usual is O(kd).

Hua *et al.* in [23] proposed an algorithm LSBF which improves approximate membership query. This algorithm uses Locality sensitive hashing functions instead of uniform and independent hash functions in bloom filter. Using LSH functions in bloom filter elements are stored locally sensitive thus can be search quickly. To reduce false positive they use an extra small size bloom filter which verifies the AMQ query result.

Jain *et al.* in [24] designed an algorithm for similarity detection for filtering the search result. This algorithm has three steps, in first step they use CDC to extract document features, in second step they use those features as set to create bloom filter and in third step bloom filters are compared if two bloom filter share more number of 1's, they are considered as near-duplicate.

WenMei Ong *et al.* have shown the performance of using parallel bloom filter over serial bloom filter for string searching algorithms [25]. The parallel bloom filter algorithm has been designed using CUDA parallel computing platform. The algorithm

generates a bit table by segmenting the input string list into block of words. The algorithm shows the significant improvement over serial bloom filter implementation.

Kyung Mi Lee and Keon Myung Lee in [26] proposed an algorithm to reduce false negatives in local sensitive hashing. To lessen the false negative rate, the items are allowed to be hashed in multiple buckets if the item is at the extreme boundary of the bucket. The algorithm results in better accuracy but it at the cost of computation and memory space.

David Gorisse *et al.* [27] included Chi^2 distance for approximate nearest neighbor search for high dimensional data. For this they have defined different hash functions. The algorithms performance is better than the original LSH on image and video data.

Kang Ling, Gangshan Wu in [28] have propounded a new frequency based LSH scheme known as FBLSH where a p-stable distribution function is used as hash function and frequency threshold number is set. Items which collide more than the threshold are considered as nearest neighbor. The scheme is space efficient but it fails to address the time complexity issue.

The shingling algorithm by Broder et al.'s [20] and random projection based approach by Charikar's [29] are studied as for determining the near-duplicate web pages. The renowned web search engines developed and used both of these algorithms and are compared on a very large scale, namely on a set of 1.6GB distinct web pages, the outcomes of which have shown that neither algorithm can determine near-duplicate pairs on the same website, but at the same time, both algorithms attain high precision on distinctive websites. Charikar's algorithm obtains a better precision of .50 in comparison to 0.38 for Broder et al.'s algorithm in finding near-duplicate pairs on different sites. A combined algorithm is also proposed which attains precision 0.79 in finding near duplicates.

In [30], Malcolm Slaney described the various optimization techniques in web scale search for nearest-neighbor retrieval. Most of the algorithms use LSH as the fundamental concept for probability to find nearest neighbor element of query. Here, an algorithm is described which also uses LSH and optimizes the parameters. One histogram is used for characterizing the distributions of distances to a point's nearest neighbors and the second histogram for characterizing the distance between a query and any point in the

data set. The LSH parameters and LSH index are returned which meets performance goal with minimum computational cost. LSH was divided into two parts in first part (a) Data is indexed and in second part (b) Search operation is performed for to find the similar records of a query point.

In [31], A. Stupar proposed an approach for implementation of locality sensitive hashing termed as RankReduce, which can be employed for similarity search on high dimensional data, on MapReduce infrastructure. In this method, the MapReduce and LSH characteristics need to be utilized at the equivalent time for achieving tremendous accuracy and exceptional performance. MapReduce is generally not employed for query processing but to process huge volume of data in an online fashion.

In spite of decades of effort, the current solutions proposed are far from satisfactory; in fact, for large dimensional data (in theory or in practice), they contribute meager enhancement over linear similarity search, where each and every pair of items is compared. So there is need to minimize the time complexity of similarity search for high dimensional data.

Chapter 3

Problem Statement

Similarity search is one of the fundamental problem in many fields of computer science like machine learning, data mining, plagiarism detection and recommender systems. The linear search is performed in case of randomly stored data and partitioning search is performed in case of already sorted data, but both of these cases take a lot of time, making these algorithms unsuitable for unstructured and high dimensional data. Researchers normally prefer to apply heuristic techniques for handling such type of unstructured data. One such technique is LSH which takes the advantage of the idea "similar items maintain similar distance even after passing through mapping".

3.1 Gap Analysis

- **Execution time complexity of high dimensional data:** The partitioning algorithms perform well for less dimensional data say 10 to 20, but as the dimensions increase, the execution time complexity of these algorithms increase exponentially. In other words, these algorithms are not efficient for high dimensional data.
- **Space complexity:** Space complexity needs to be decreased for similarity search in case of high dimensional data.
- **Accuracy:** Most of the similarity search techniques use heuristic methods which fail to provide satisfactory level of accuracy. In most cases, these techniques produce erroneous results (false positive and false negative), so the error rate needs to be reduced.

3.2 Objectives

- To study, analyze and explore the existing similarity search techniques for high dimensional data.
- To propose a technique for improvement in time complexity of similarity search for high dimensional data.

- To test and validate the proposed methodology with proposed algorithm for similarity search.

3.3 Methodology

Major aim of the thesis is to improve time complexity of similarity search for high dimensional data. Methodology followed is:

- Locality Sensitive Hashing is used for finding candidate pairs so that only subsets of items are compared.
- Bloom Filter is used to improve space complexity and it also provides constant search time for membership test.
- This algorithm has been tested for four text files. Shingles sizes for those files are taken as 5, 6, 7, 8, 9 and 10.
- Proposed algorithm has also been tested for different file sizes. The various file sizes of the four files were taken as 4KB, 16KB and 55KB.

Chapter 4

LSH and Bloom Filter

4.1 Nearest Neighbor Search

Nearest neighbor search (NNS) is an optimization problem to find nearest neighbors in metric spaces. It is defined as follows:

Definition: "Given a set P of n points in a d-dimensional metric space M = (X, d) and a query point q $\in$ X, find the point p $\in$ P closest to q" [32].

NNS is an optimization problem to locate the most similar items in a given dataset. Similarity or closeness is usually expressed in terms of a dissimilarity function which says "The less similar are the objects the larger are the function values". Due to an application of assigning to a residence the nearest post office, nearest neighbor search problem was also known as post-office problem [32] and direct generalization of this problem is a k-NN search, where the k closest points are find out. Generally, M is a metric space and dissimilarity is usually expressed as a distance metric, which is symmetric and also satisfies the triangle inequality. More specifically, M is taken as the d-dimensional vector space where dissimilarity is measured using various functions of the Minkowski family such as Euclidean distance, Manhattan distance etc. However, there can be arbitrary dissimilarity function.

4.2 Approximate Nearest Neighbor Search

For some applications, accuracy is not as important as the speed. Approximate Nearest Neighbor (ANN) is a searching technique which exploits this freedom of accuracy. Ciaccia and Patella introduced an accuracy parameter 'ε' and a confidence parameter 'δ' which motivates the following definition of ANN:

Definition: "Given a set P of n points, in a d-dimensional metric space X, given an accuracy parameter ε and a confidence parameter δ $\in$ [0, 1). Design a data structure that, given a query point q $\in$ X returns a point p' $\in$ P which has a distance of at most

1+ ε times the distance of q to its nearest neighbor in P with probability at least 1-δ. For δ = 0 it is called (1+ ε)-NN, for δ > 0 (1+ ε, δ)-NN" [33].

4.3 Locality Sensitive Hashing

In general hashing, closed (near) items may be hashed in different locations after hashing, but in case of Locality Sensitive Hashing items maintain their closeness even after hashing (mapping), as shown in figure 4.1

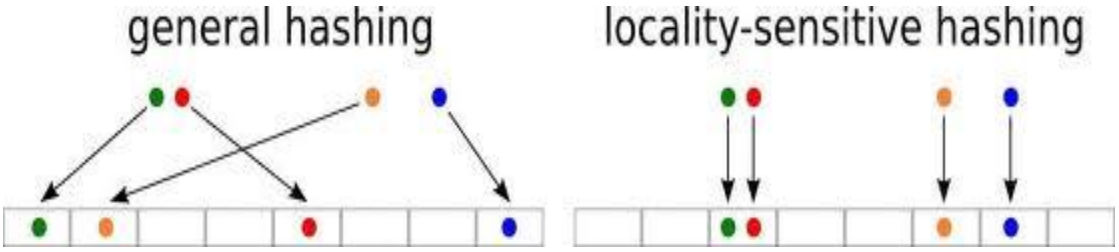

Figure 4.1 Difference between general hashing and locality sensitive hashing [39]

In LSH, candidate pairs are those pairs which hash to the same bucket when banding technique is applied and comparisons are performed only on candidate pairs rather than comparing every pair like in linear search. If the requirement is to find exact match then one of the techniques to process data like Map Reduce, Twitter storm etc. can be used. These techniques are based on parallelism so result in time reduction. But these methods require extra hardware. Sometime exact result is desired in many cases we only want pairs that are most similar. The similarity level is defined by some threshold. In these cases the desired result is called Nearest Neighbor Search or LSH.

To use LSH in various applications, one needs to develop it according to the domain defined for that application. When finding similar documents, a family of functions called Minhash functions is combined with banding technique to separate pairs at low distance from the pairs at large distance [34]. There are three criteria defined for a family of functions:

- The close pairs should be made as candidate pairs rather than distant pairs.
- These functions must be statistically independent from each other so that probability of two or more functions will hash to same result can be determined.
- These must take lesser time than the time required for comparing all pairs.

Another important consideration is that false negatives and false positives should be avoided as far as possible. Further, the combined function must take very less time than the number of pairs.

Let us assume d1 and d2 are two distance parameter (d1<d2) according to some distance measurement d and F is family of functions. F is called (d1, d2, p1, p2)-sensitive if for all f belongs to F;

Probability of f(x) =f(y) is

$$\begin{array}{ll} at\ least\ p1 & if\ d(x,y) < d1 \\ at\ most\ p2 & if\ d(x,y) > d2 \end{array}$$

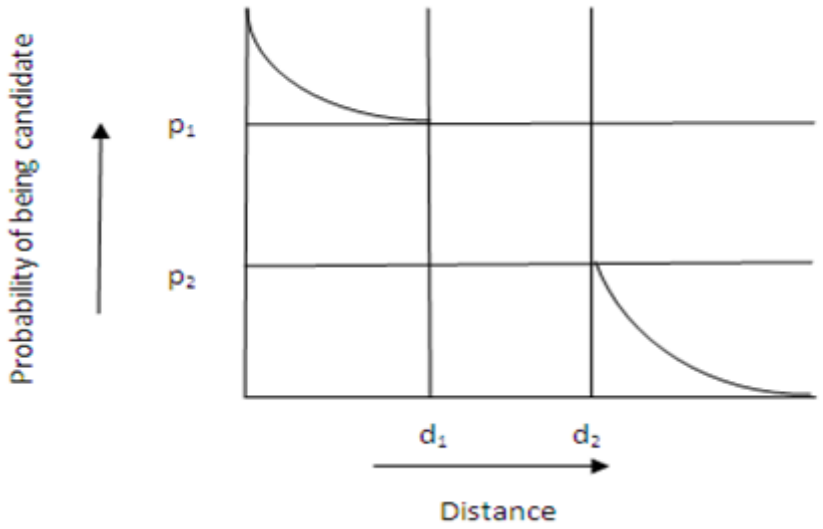

Figure 4.2: Behavior of a (d1, d2, p1, p2)-sensitive function

Figure 4.2 depicts a graph in which x-axis represents distance and y-axis represents the probability of being candidate pairs. If the distance of points is in between d_1 and d_2 then the probability cannot be defined. d_1 and d_2 can be close to each other but the problem is that then p_1 and p_2 will not be distinguishable[11].

4.4 Bloom Filter

Bloom Filter, a probabilistic data structure that uses multiple hash functions to store data in a large bit array was introduced in 1970 by Burton H. Bloom [35]. It is mainly used for

membership queries when dealing with large data sets. It reduces O(n) search time to constant time. The search accuracy depends on the size of Bloom Filter (m) and number of hash functions(k) and the major advantage of multiple hash functions is that search accuracy is improved. This approach is very effective when speed matters more than space. Suppose we have a set U having n elements, which are to be hashed using Bloom Filter. The size of the Bloom Filter is m. Size of Bloom Filter is usually taken larger than the size of data to be stored to avoid collisions. K, the number of hash functions, depends on the size of data to be stored and the size of the bit array (bloom filter).

Mathematically:

$$k = \left(\frac{m}{n}\right) ln2 \quad (3)$$

Each hash function returns a value between 1 to m. To hash an element in BF, it is hashed by all k hash functions, and all the bits corresponding to these k values are set to 1. If any of the k value is already set as 1, it is left unchanged. When checking for membership, the element is hashed to all k hash functions and the bit array is checked for all k values, if all of the k values are 1, the element is a member of the set. While implementing Bloom filter, there are chances of false positives, where it may show that an element is a member of set in U even if, it is not. From the analysis in [35,36], if the cardinality of set U is n, and the size of Bloom filter is m the optimal value of k that minimizes the false positive probability, p^k, where p denotes that probability that a given bit is set in the Bloom filter, is $k = \left(\frac{m}{n}\right) ln2$.

For example, figure 4.3 illustrates bloom filter which uses k number of hash functions to store n number of items in m bit array.

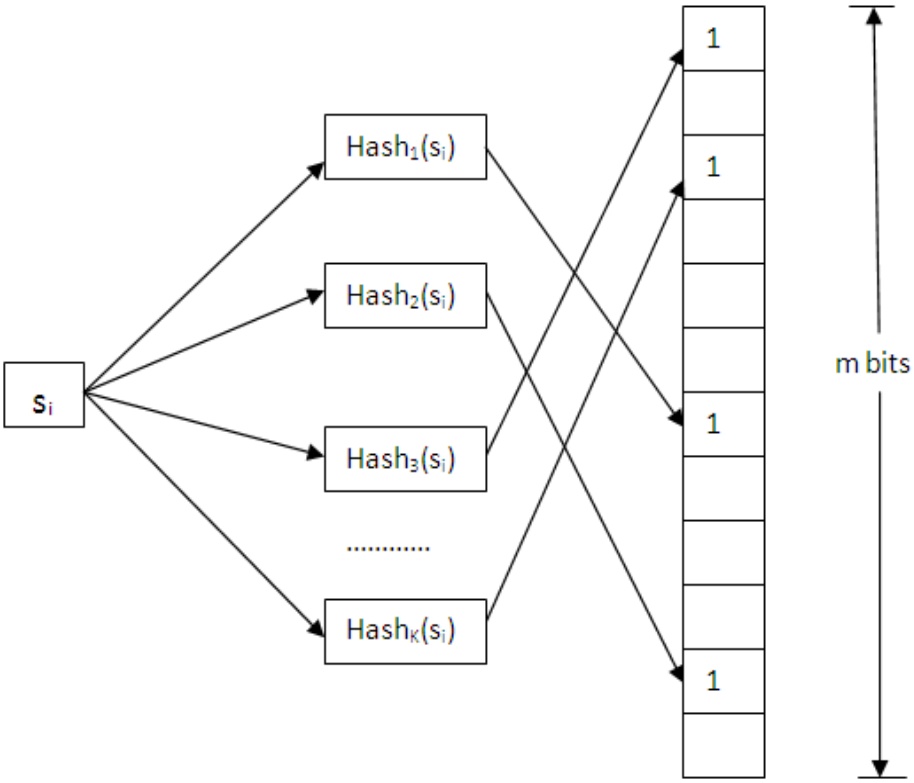

Figure 4.3: Bloom filter using k Hash Functions [37]

The Bloom filters have the property that the false positives can be reduced. The larger the size of bloom filter, less are the false positives and smaller the size, more are false positives. The false positive rate is approximately $(1-e^{-kn/m})^k$, with *n* number of elements which are inserted, various values of *m* and *k* are used to construct the filter for the requisite application. Therefore, the more hash functions slows the bloom filter and it also fills up quickly. However, if few hash functions are used in bloom filter, it results in more false positives.

Therefore, given the values of *m* and *n*, function can be used to choose the optimal value of $k = \left(\frac{m}{n}\right) ln2$

So, the steps used in choosing the size of bloom filter are:

1. For a value of *n* (number of items)
2. A value for m is selected

3. The suitable value of *k* is calculated
4. The error rate is calculated for the selected *n*, *m* and *k* values. If the error rate is not acceptable, return to step 2 and the value of *m* is changed;

The insertion and membership testing of an item are of complexity *O(k)* in the m bits bloom filter and with k number of hash functions. Every time an element is added to the set or checked for set membership, the element needs to run through the *k* hash functions.

Chapter 5

Design Techniques

5.1 Set and Operations

A set is a (unordered) collection of objects. Sets are usually represented in curly braces, each member separated by comma (,).

- The size of a set also called *cardinality* of a set is the number of elements it has.
- There are mainly three operations which are defined for *sets*, Union, Intersection, and Difference.

Union of two sets A and B is denoted A ∪ B. The result of union of two or more sets is also a set which has all the elements contained by set A *or* set B. Union satisfies symmetry property i.e. union of A and B is same union between B and A.

Intersection of two sets A and B is denoted A ∩ B. The result of union of two or more sets is also a set which has all the elements contained by set A *and* set B. Intersection satisfies symmetry property, i.e. intersection of A and B is same union between B and A.

The *difference* of set A with set B also results in a set having all those elements from set A which are not the member of set B. It is to be noted that set difference does not hold symmetry i.e. A-B is not same as B-A.

A distance between two objects has the following properties:

- It is small if A and B are close enough,
- It is large if they are far from each other,
- It is 0 if they are same,
- It is in the range [0, ∞]

Similarity between two objects is defined as how two objects are similar to each other S(A, B) has the following properties:

- It is large if the objects A and B are close to each other,

- It is small if they are dissimilar,
- It is (usually) 1 if they both are the same,
- It is in the range of [0 1].

The relation between distance and similarity of two objects A and B is

d(A,B) = 1 - s(A,B).

5.2 Jaccard Similarity

Jaccard Similarity [11] of two sets is the ratio of the size of the intersection to the size of the union of the two sets. Let there are two sets, set A and set B, than mathematically the Jaccard Similarity

J-SIM (A, B)

$$\text{J-SIM}\ (A, B) = \frac{|A \cap B|}{|A \cup B|}$$

For example, if set ***A*** has elements {2, 3, 4, 5, 6} and set *B* has elements {1, 3, 4, 5, 6, 7} the J-SIM (A, B) = 4/7.

5.3 Shingle's hashing

The hash function should be able to hash strings of size k to some number of buckets and use the resulting bucket number as the shingle. This hashing scheme will save lots of space in actual implementation.

5.4 Shingles at Character level

Shingle sets can also be formed at the character level, in which parameter k happens to be the number of characters instead of words.

5.5 Shingles Model

There are some considerations when designing shingle's model:

- White space- Should spaces be considered.
- Capitalization- Case sensitivity should be considered.
- Punctuation- punctuation symbols are the part of shingles or not

- Characters/ Words- character based shingles or word based shingles.
- The size k- K should be chosen as to maximize the collision. Size k depends on the types of documents. For example, for e-mails, the size k=5 is appropriate, and for research paper k=9. For other kind of documents (Blogs, news articles, web pages) k lies between 5 to 9.
- Count replicas- Typically bag of words counts replicas, but shingling does not.
- Stop words – The words which are very common for the document and repeated too much, are called stop words. In English words like a, an, i, he, you, for, the, to, and, that, it, is etc. are very common. Usually these are not considered while shingling.

5.6 Characteristic Matrix

We represent the sets in characteristic matrix say M. The size of this matrix is (m x n) where m is the number of rows and n is the number of sets. The columns of the matrix are the sets, while the rows store the elements of universal set. The entry Mij= "1", if ith shingle is present in jth set, otherwise Mij="0", where 0<=i<=m, and 0<=j<=n. Some hash functions are defined which are much less than number of row. All these hash functions generate some permutations of present rows. These hash functions will we used to create Minhash signature matrix.

Let there are five documents D1,D2,D3,D4 AND D5 and their shingles sets are DS1 {A, F, G}, DS2 {A, B, C}, DS3 {A, E, F, G}, DS4 {A, B, C, D, E}, and DS4 {C, D, E, F, G} respectively.

Shingles set {A, B, C, D, E, F, G} after hashing every shingle gets a bucket number let bucket numbers are {0, 1, 2, 3, 4, 5, 6} respectively. Applying rules of "Matrix representation of sets" resulting matrix will be as in Table 5.1.

Table 5.1: Matrix representation of document sets (Characteristic matrix)

Shingles	DS1	DS2	DS3	DS4	DS5
0	1	1	1	1	0
1	0	1	0	1	0
2	0	1	0	1	1
3	0	0	0	1	1
4	0	0	1	1	1
5	1	0	1	0	1
6	1	0	1	0	1

5.7 Minhashing

Minhashing is a type of LSH techniques in which independent permutations of a given column are used for finding the similarity of items (sets).This technique was developed by A. Broder and was first used in search engine (AltaVista) for finding and removing the duplicate web pages from the search result. It can also be applied for clustering, in which documents can be clustered based on the similarity of their set of words present in documents.

5.8 Signature Matrix

Signature matrix (SM)[11] consists of 'h' rows that are the number of hash functions and numbers of columns are same as in case of characteristic matrix. Every element in this matrix is initialized with a large number. To create signature matrix, characteristic matrix is scanned row by row. For all the sets for which the entry is "1" in characteristic matrix, for that set the corresponding hash values are compared with the existing hash functions values in Signature matrix. If the value is found to be less it is updated in signature matrix, otherwise no changes are performed. The entries of signature matrix give us similarity almost same as in case of characteristic matrix, but it takes very less storage pace.

Table 5.2: Characteristic matrix with hash functions

Shingles	DS1	DS2	DS3	DS4	DS5	Hash1	Hash2	Hash3
0	1	1	1	1	0	1	2	3
1	0	1	0	1	0	2	3	4
2	0	1	0	1	1	3	4	5
3	0	0	0	1	1	4	5	6
4	0	0	1	1	1	5	6	0
5	1	0	1	0	1	6	0	1
6	1	0	1	0	1	0	1	2

In Table 5.2, characteristic matrix is shown combined with the three hash functions Hash1 ((sn+1) mod 7), Hash2((sn+2) mod 7) and Hash3((sn+3) mod 7), Where sn is the corresponding shingle number.

Initial Signature matrix is represented as in Table 5.3.

Table 5.3: Signature matrix

	DS1	DS2	DS3	DS4	DS5
Hash1	∞	∞	∞	∞	∞
Hash2	∞	∞	∞	∞	∞
Hash3	∞	∞	∞	∞	∞

After applying first scanning of characteristic matrix, the Signature matrix is shown in Table 5.4.

Table 5.4: After scan of first row of characteristic matrix

	DS1	DS2	DS3	DS4	DS5
Hash1	1	1	1	1	∞
Hash2	2	2	2	2	∞
Hash3	3	3	3	3	∞

In first row of Characteristic matrix all the documents except DS5 have value “1”. So for the four documents DS1, DS2, DS3 and DS4 in signature matrix, the hash values are updated with the corresponding hash values in characteristic matrix. Same procedure is applied for all remaining rows as shown in table 5.5, 5.6, 5.7.

Table 5.5: After scan of row 2/row 3 of characteristic matrix

	DS1	**DS2**	**DS3**	**DS4**	**DS5**
Hash1	1	1	1	1	3
Hash2	2	2	2	2	4
Hash3	3	3	3	3	5

Table 5.6: After scan of row 4 of characteristic matrix

	DS1	**DS2**	**DS3**	**DS4**	**DS5**
Hash1	**1**	**1**	**1**	**1**	**3**
Hash2	**2**	**2**	**2**	**2**	**4**
Hash3	**3**	**3**	**0**	**0**	**0**

Table 5.7: After scan of row 5 of characteristic matrix

	DS1	**DS2**	**DS3**	**DS4**	**DS5**
Hash1	1	1	1	1	3
Hash2	0	2	0	2	0
Hash3	1	3	1	0	1

After scanning all rows in characteristic matrix, the resulting signatures matrix is shown in Table 5.8.

Table 5.8: Signature matrix after scanning all rows of characteristic matrix

	DS1	**DS2**	**DS3**	**DS4**	**DS5**
Hash1	0	1	0	1	0
Hash2	0	2	0	2	0
Hash3	1	3	1	0	1

The signature matrix in Table 5.8 shows that documents DS1, DS3 and DS4 are similar to each other.

5.9 LSH on Signature Matrix

Signature matrix takes very less space to represent documents because it stores only signatures of the documents. But even for fewer documents it still makes too much comparisons for finding similar pairs, which approximately takes time that is ${}^{n}C_2$ ($O(n^2)$). This problem of too many comparisons of items can be solved by banding technique.

5.10 Banding Technique

In the banding technique, number of rows of signature matrix are divided into (b x r) where b represents the number of bands (figure 5.1). Each band consists of maximum r rows. For each band a hash function is defined. The hash function takes column of its corresponding band and hashes them to large number of buckets.

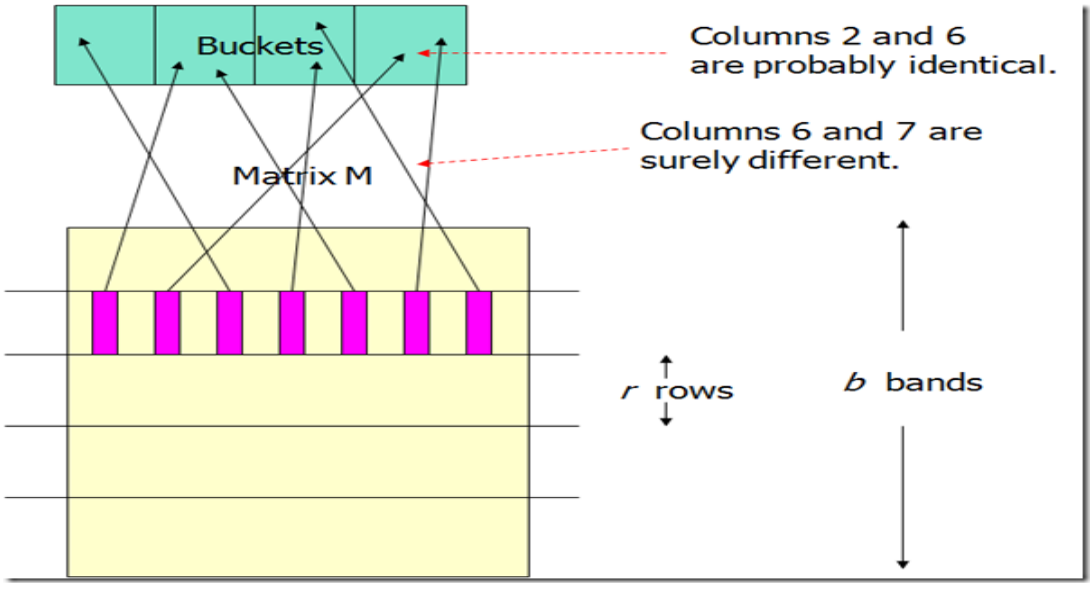

Figure 5.1 Signature matrix divided into b bands and r rows

For all the bands same hash functions can be chosen, in that case a dedicated bucket array for each band is used, so that same bucket elements would be candidate pair. The banding technique also might have some problems of false negative and/or false positive. False negative problem is when the most similar pairs are not hashed in same bucket.

False positive problem is when the dissimilar pairs are hashed to same bucket. Although the probability of false negatives and false positives is considered to be very less, the problem of false positive can be eliminated by direct comparing those candidate pairs.

5.11 Analysis of the Banding Technique

Suppose there are **b** bands of **r** rows each, and suppose that a particular pair of documents have Jaccard similarity **s**. Then the probability that Minhash signatures for these documents lie in any single row of the signature matrix is **s**.

- The probability that the signature match in every row of a particular band is $\mathbf{s^r}$.
- The probability the signatures do not match in at least one row of a particular band is $\mathbf{1–s^r}$.
- The probability that the signatures do not match in all rows of any of the bands is $\mathbf{(1 – s^r)^b}$.
- The probability that the signatures match in all the rows of at least one band, and therefore become a candidate pair, is $\mathbf{1 − (1 – s^r)^b}$.

This function has the form of an S-curve as shown in figure 5.2, regardless of the chosen constants b and r. It can be clearly depicted that the threshold, the value of similarity s at which the rise becomes steepest, is a function of b and r. An approximation to the threshold is $\mathbf{(1/b)^{1/r}}$.

s	1-(1-S^r)^b					Rows-Per Ban	Bands
0.2	0.0023					6	36
0.3	0.02591						
0.4	0.13736						
0.5	0.43274						
0.6	0.82094						
0.7	0.98896						
0.8	0.99998						

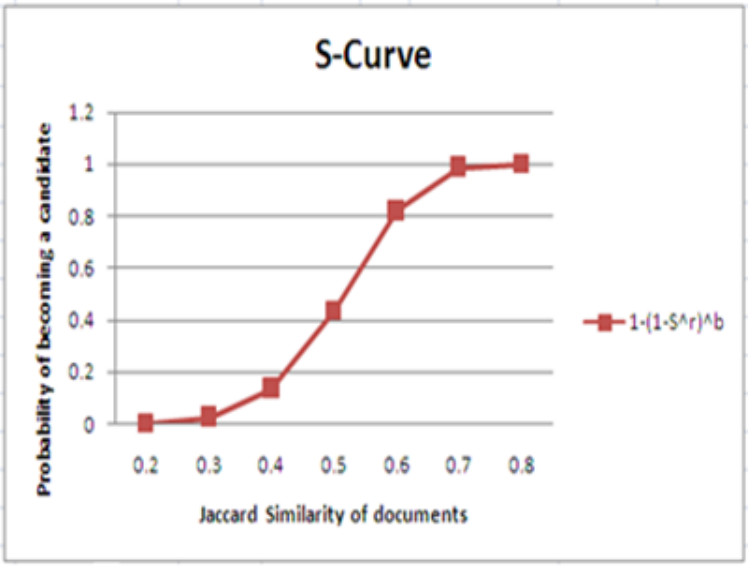

Figure 5.2: S-Curve

Chapter 6

Implementation and Results

In creation of characteristic matrix, searching of an element in union set of shingles take too much time. This time is O(n) if shingles are not sorted. If union set of shingles are sorted in preprocess step then it takes O(logn) time. In this work, the use of a probabilistic hashing Bloom Filter is proposed which makes the search time in shingles set constant. The detailed steps of the algorithm and the use of bloom filter are described hereby.

6.1 Algorithm proposed in [11] for Similarity search

Input: Set of files

Output: Most similar files grouped together

Step1: Select all files

Step2: Convert text files into shingles set

Step3: Store shingles into bloom filters for each file, and union of all shingles set into a large array.

Step4: Create characteristic matrix

Step5: Create signature matrix

Step6: Divide signature matrix into of n rows into b band and in every band r rows

Step7: Hash the columns of signature matrix in a large bucket by using different hash function for different band Use of bloom filter for storing the shingles improves the search time

6.2 Proposed efficient algorithm of finding similar documents

Bm_1, Bm_2,……..Bm_n are bloom filters which store shingle in form of 0's and 1's.

Shingle's hash table has all shingles and every shingle has index number

CM is characteristic matrix whose rows are shingles index number and columns are sets or bloom filters. Entry for every CM(r,c)=1 if r shingle present in Set c, otherwise

CM(r,c)=0. Searching of r shingle in set c takes Log(n) time if set is used and search is completed in constant time if bloom filter is used.

Working of bloom filter is based on hashing an item in multiple locations at the same time and it takes k hash function to generate k hash values of an item, as the size of increases chances of false negative reduces.

SM is a signature matrix that stores the some useful information from CM but gives almost same information as CM of course not exact same, but still meaningful to find similarity. SM is created by using CM and some k hash function. Number of hash functions are depends upon the accuracy required. These hash function gives one of the permutation of the shingle's index number and every hash function gives different permutation.

Step1: Select all files

Step 2: Convert text files into shingles

Step 3: Store shingles into bloom filters of each file. Hash union of all shingles into a large bucket.

Step 4: Create characteristic matrix

Let CM[n, m] is characteristics matrix with n rows and m columns, where rows are representing shingles(Si) and columns are representing bloom filters(BFj) *i.e.*
S_i, where i= 1 to n [Si is a particular shingle]
BF_j, where j =1 to m [BFj is a bloom filter of shingles of a particular file]

```
For i = 1 to n
    For j= 1 to m
      If(Si in BFj) [if shingle i in BF j]
          then CM[i, j]=1
      else
          CM[i,j]=0
      End if
    End For
```

End For

Step 5: Create signature matrix

Step 6: Divide signature matrix into of n rows into b band and in every band r rows

Step 7: Hash the columns of signature matrix in a large bucket by using different hash function for different band.

6.3 Jaccard Similarity of Documents

The similarity of two documents is compared by using Jaccard similarity and the similarity is calculated by removing the stop words. Stop words are removed because they do not provide meaningful result. The result of similarity lies between 0 and 1. It tells how much two documents are similar. If the similarity of two documents is 1 it means both the documents are identical, on the other hand if similarity is 0, it means that documents are entirely different. Jaccard Similarity of two text files, with and without removing stop words is calculated as shown in figure 6.1 and figure 6.2.

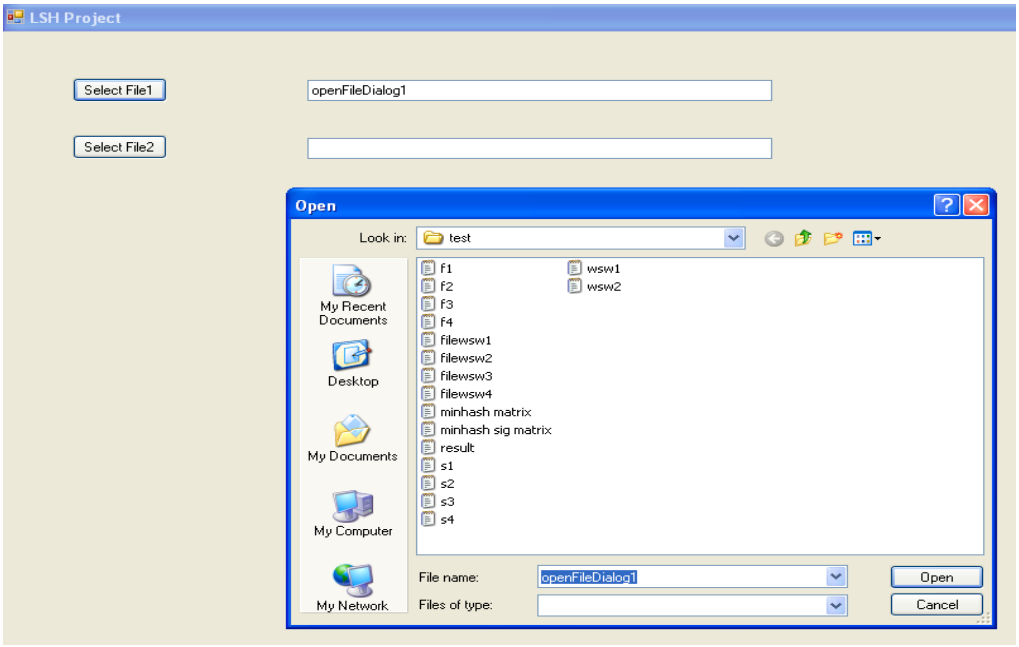

Figure 6.1: File selected by user

Files name and the shingle size are the input of this program. In figure 6.1 user selects the files needed to be compared by clicking on the Buttons “Select File1” and “Select File2”.

After selecting both the files, user needs to input the shingle size according to the types of documents in consideration (say 5). The Jaccard similarity can be calculated by clicking on the button captioned "Jaccard Similarity". The output of the program is shown in Figure 6.2. In front of label "with stop word" the similarity of selected documents is 0.8333, and "without stop word", it is only 0.625. The result of the program also gets saved in a file named as *result.txt* for later comparisons.

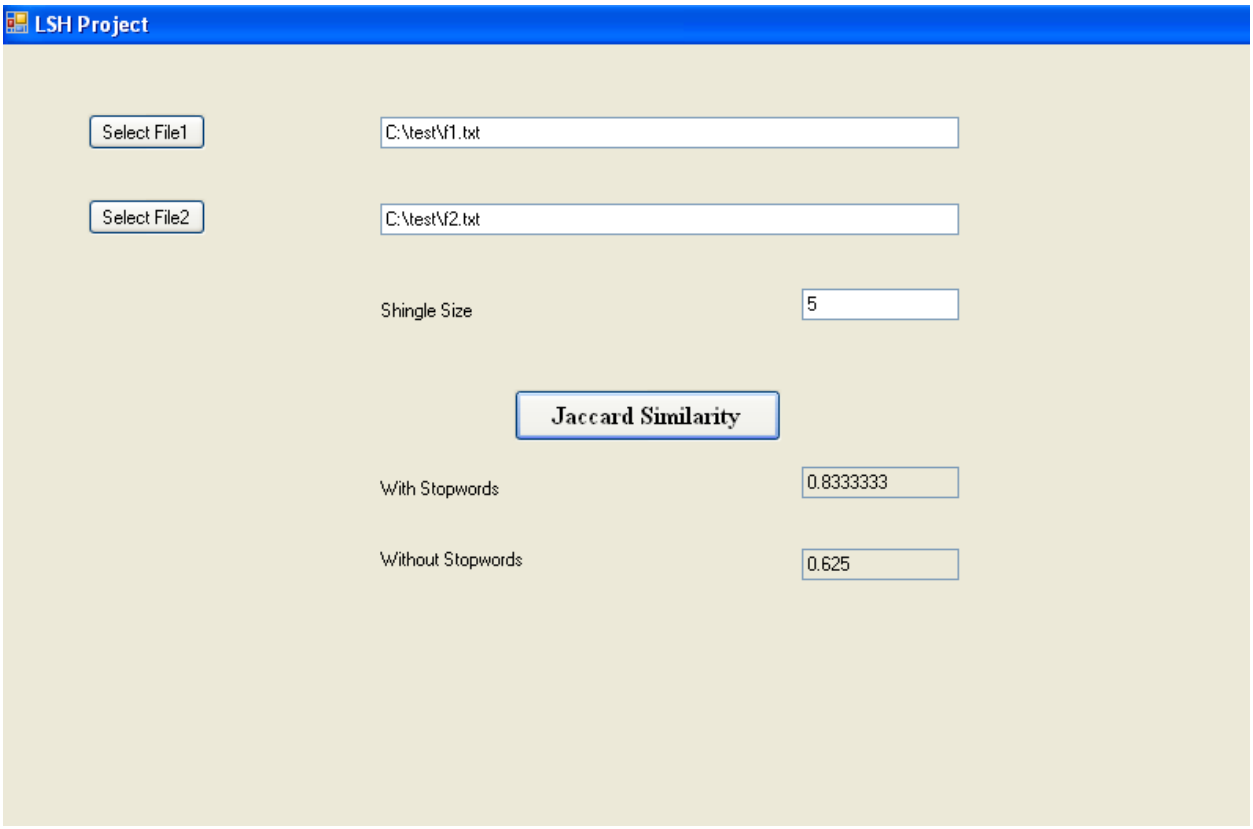

Figure 6.2: Jaccard Similarity of selected files

Contents of file result.txt are shown in Figure 6.3, it stores file name, shingle size, Jaccard Similarity of files with and without stop words.

File Names	Shingle Size	JS with Stop words	JS without Stop words
(C:\test\f1.txt, C:\test\f3.txt)	3	0.07692308	0.06666667
(C:\test\f1.txt, C:\test\f3.txt)	5	0.1111111	0.09090909
(C:\test\f1.txt, C:\test\f3.txt)	6	0.1428571	0.1111111
(C:\test\f1.txt, C:\test\f3.txt)	3	0.07142857	0.0625
(C:\test\f1.txt, C:\test\f3.txt)	4	0.08333334	0.07142857
(C:\test\f1.txt, C:\test\f3.txt)	5	.1	0.08333334
(C:\test\f1.txt, C:\test\f3.txt)	6	0.125	0.1
(C:\test\f4.txt, C:\test\f3.txt)	6	0.1428571	0.1111111
(C:\test\f4.txt, C:\test\f3.txt)	5	0.1111111	0.09090909
(C:\test\f4.txt, C:\test\f3.txt)	4	0.09090909	0.07692308
(C:\test\f4.txt, C:\test\f3.txt)	3	0.07692308	0.06666667
(C:\test\f3.txt, C:\test\f2.txt)	3	0.07692308	0.06666667
(C:\test\f1.txt, C:\test\f2.txt)	5	0.8333333	0.625

Figure 6.3: Contents of result.txt file

6.4 Characteristics matrix of documents sets and Signature Matrix

This program is used to find similar documents in a given collection of documents by using the techniques S*hingling* and M*inhashing*. In Figure 6.4 program interface is shown in which user can select multiple files. Here, four files from different news papers for same article are selected.

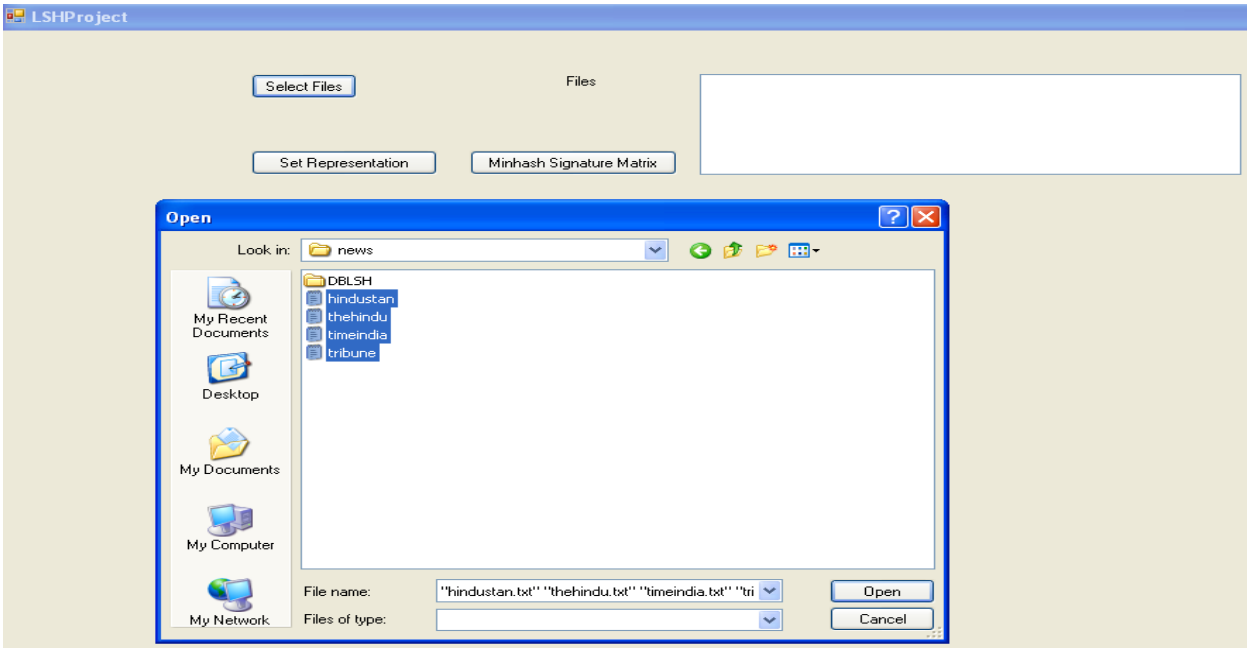

Figure 6.4: Various files selected by user

6.4.1 Characteristics matrix of documents set

A list box front of label captioned as “Files” shows the selected filenames. A button captioned as “Set Representation” is used to show Matrix representation of Documents set and another button captioned as “Minhash Signature Matrix” is used to show signatures of documents sets. When user clicks on button “Set Representation” a list is opend in which first column represents the shingle numbers, second column represents shingles of selected size, and next five columns are hash functions and remaining four columns represent the entries in files according to the rule that if the shingle is present in that file then entry is 1, 0 otherwise. The contents of this listbox can be saved by clicking on button “characteristics matrix” (Figure 6.5).

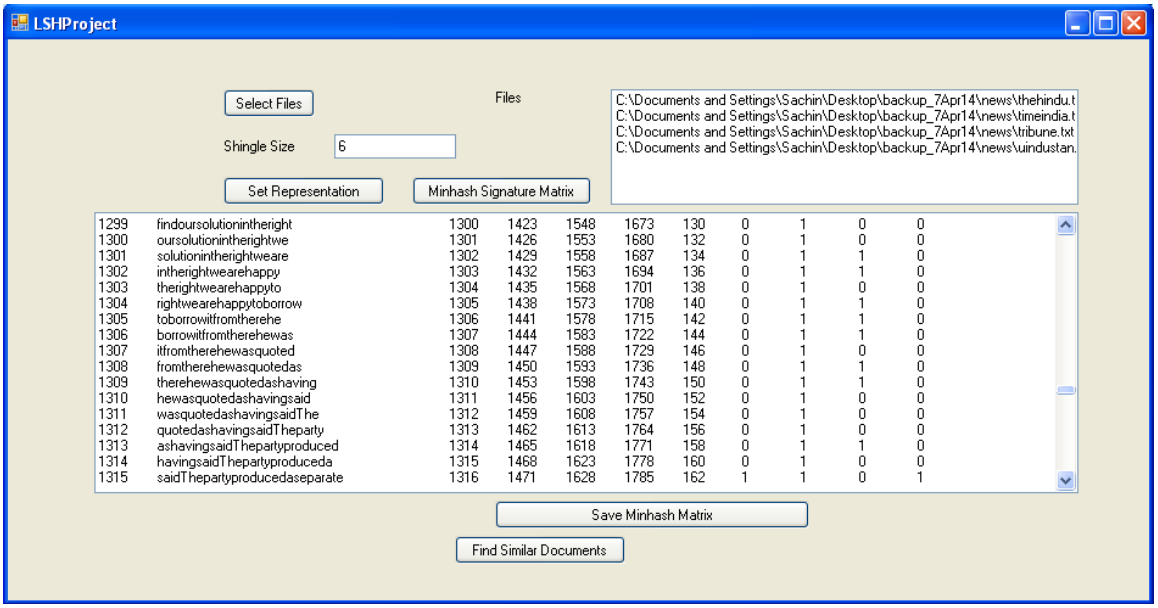

Figure 6.5: Characteristics matrix of the selected files

6.4.2 Signature Matrix

In Figure 6.6, signatures of all four files are shown. On the basis of these signature, the similarity of documents can be determined. The similarity calculated from these signature is almost same as if we compared whole file. From these signatures it can easily be seen that File1 and File2 are identical.

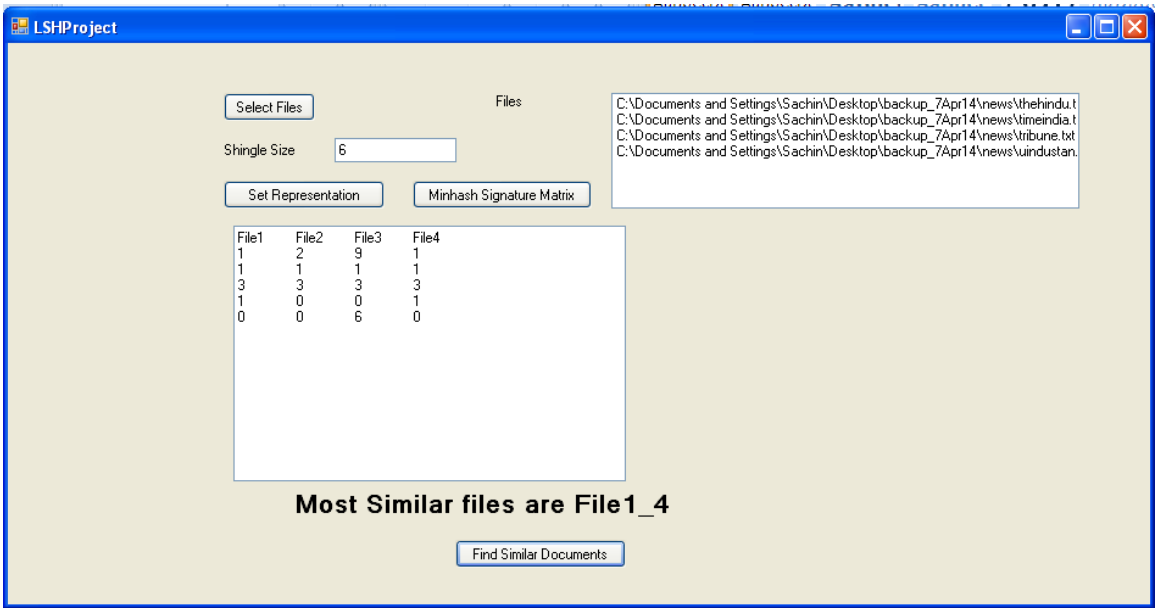

Figure 6.6: Signature matrix of characteristic matrix

6.5 Results and Analysis

This algorithm has been run for four text files which contain news on same context in different newspapers. Shingles sizes for those files are taken as 5, 6, 7, 8, 9 and 10. Figure 6.7 provides a comparative analysis of existing and proposed algorithm. It clearly shows that the proposed algorithm has constant search time complexity.

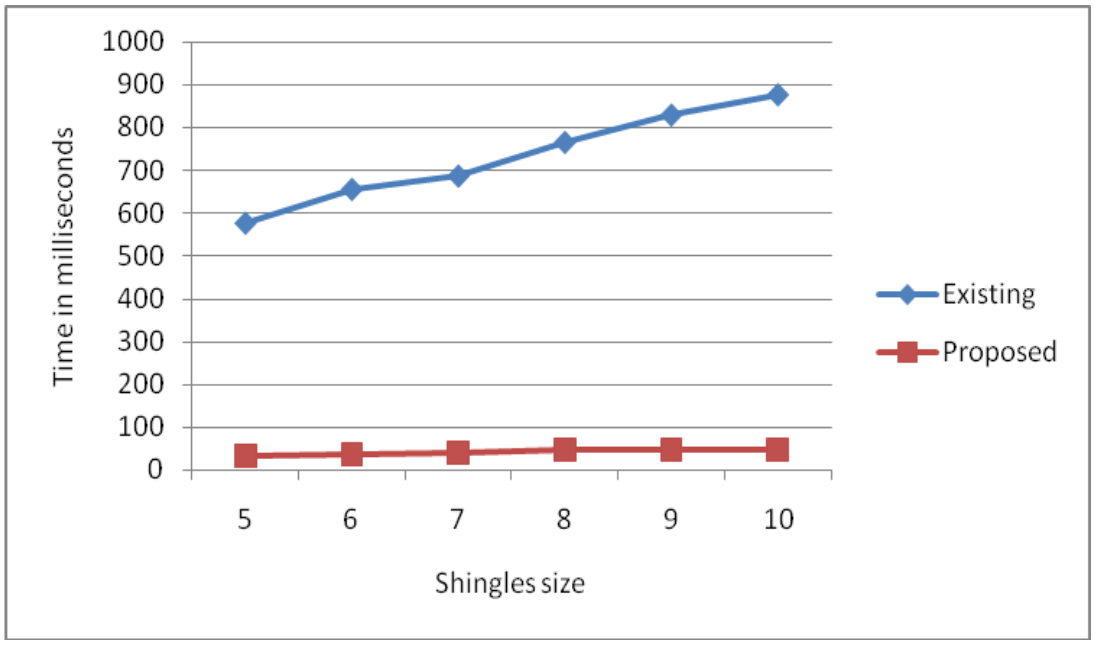

Figure 6.7: Comparison of execution time for various shingles sizes

Table 6.1 gives the comparison of the execution times of existing and proposed algorithms with different file sizes. The Table 6.1 shows that the execution time of the proposed algorithm is considerably reduced even if the size of the file increases. Figure 6.8 gives the graphical representation of the execution time with different file sizes.

Table 6.1: Time (in milliseconds) taken by various files in existing and proposed algorithms

Total File Size	**Execution time**	
	Rajaraman *et al.* [11]	**Proposed**
4KB	(46+31+31)/3=36	(46+31+31)/3=36
16KB	(656+671+671)/3=666	(400+348+328)/3=369
55KB	(1000+1200+1133)=1111	(609+498+593)/3=567

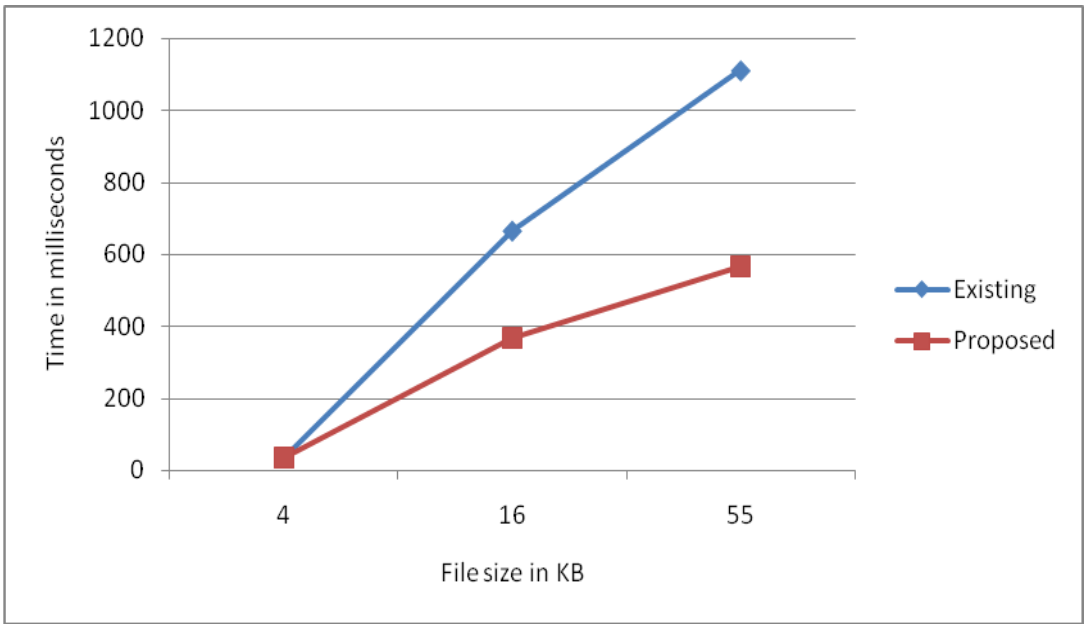

Figure 6.8: Representation of the execution time with different file sizes

Thus, the proposed algorithm reduces the time complexity of the similarity search and the execution time also gets reduced when various file sizes are taken into consideration.

Chapter 7

Conclusion & Future Scope

7.1 Conclusion

In this work, similarity search problem has been solved by using Locality Sensitive Hashing and Bloom Filter. Locality Sensitive Hashing has been build upon signature matrix and banding technique where signature matrix is created by using characteristic matrix and some hash functions. Characteristic matrix is build by using shingles of documents and sets of shingles of each document, and hash functions are used to generate some random permutations of shingle number. In characteristic matrix, searching of a shingle linearly depends on file size, i.e. O(n), which is significantly large time when processing large files. In this work, an algorithm has been proposed which uses Bloom Filters for storing shingles, resulting in constant search time and comparative nalaysis has been provided with the existing algorithm.

7.2 Summary of Contribution

The similarity search technique used in this work is applicable to majority of items like image, graph, network, video, text etc., but this is specially designed for text similarity search. In this work Bloom filter and locality sensitive hashing are effectively used to find the ε-approximate similar items. Here ε is the error in terms of false negative and false positive. One of the commendable features of this work is the flexibility of system to calculate similarity between set of items in various prominent computer science applications like recommender system, news extraction, plagiarism detection and many other applications of data mining and machine learning.

7.3 Future scope

In this work, simple hash functions are used in Bloom Filter and in Locality Sensitive Hashing. Both accuracy and speed depends mainly upon those hash functions. In future, this work can be made more effective in terms of speed and accuracy by using some fast data (problem) specific hash functions. The problem of false positive due to bloom filter

and the problem of false negative and false positive due to LSH will also be minimized by using better hash functions. The bloom filter size and number of hash function can be selected by using some statistical and probabilistic methods.

Apart from selecting good hash functions, this work can also be extended in the following two directions:

- Firstly, this algorithm can be utilized for parallel processing environment such that shingling of documents can be distributed in parallel and multiple files can be selected for similarity search.
- Secondly, this work can also be extended to for finding semantic similarity in text documents i.e. to find the similarity between text files which are semantically similar.

References

[1] J. R. Smith, “Integrated Spatial and Feature Image Systems: Retrieval, Analysis, and compression”, PhD thesis, Graduate School of Arts and Science, Columbia University, 1997.

[2] “Shingles and near-duplicates” [online] available at http://nlp.stanford.edu/IR-book/html/htmledition/near-duplicates-and-shingling-1.html, last accessed on 30th April, 2014.

[3] A. Bosch, X. Muñoz, and R. Martí, “Which is the best way to organize/classify images by content?”, Image and Vision Computing, vol. 25, no. 6, pp. 778–791, 2007.

[4] U. Manber, “Finding similar files in a large file system”, Proc. USENIX Conference, 1994.

[5] C. M. Bishop, “Pattern Recognition and Machine Learning”, Springer-Verlag, Secaucus, NJ, USA, 2006.

[6] J.L. Bentley, “Multidimensional binary search trees used for associative searching”, Comm. ACM, pp. 509-517, 1975.

[7] P. Ciaccia, M. Patella, and P. Zezula, “M-tree: An efficient access method for similarity search in metric spaces”, In PVLDB, pages 426–436, 1997.

[8] G. Beskales, M. A. Soliman, and I. F. IIyas, “Efficient search for the top-k probable nearest neighbors in uncertain databases”, PVLDB, 1:326–339, August 2008.

[9] T. Seidl and H. P. Kriegel, “Optimal multi-step k-nearest neighbor search”, In SIGMOD, 1998.

[10] A. Awekar and N. F. Samatova, “Fast Matching for All Pairs Similarity Search”, IEEE/WIC/ACM International Joint Conferences on Web Intelligence and Intelligent Agent Technologies, WI-IAT '09, 15-18 Sept. 2009, Milan, Italy, pp. 295 – 300, 2009.

[11] A. Rajaraman, J. Ullman, “Mining of Massive Datasets”, Cambridge University Press, December 30, 2011.

[12] P. Indyk and R. Motwani, “Approximate nearest neighbor: towards removing the curse of dimensionality”, Proc. Symposium on Theory of Computing, 1998.

[13] R. Krauthgamer, and J. Lee, “Navigating nets: simple algorithms for proximity search”, Proc. of 15th annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, pp. 798-807, 2004.

[14] A. Beygelzimer, S. Kakade, and J. Langford, “Cover trees for nearest neighbours”, ICML, ACM, New York, USA, pp. 97-104, 2006.

[15] A. Guttman, “R-trees: A dynamic index structure for spatial searching”, SIGMOD, pp.47-57, 1984.

[16] N. Katayama, and S. Satoh, “The sr-tree: an index structure for high-dimensional nearest neighbor queries”, SIGMOD, pp. 369-380, 1997.

[17] M. Datar el al., “Locality-sensitive hashing scheme based on p-stable distributions”, Proc. ACM Symposium on Computational Geometry, 2004.

[18] G. Junhao et al., “Locality-Sensitive Hashing Scheme Based on Dynamic Collision Counting”, ACM, 2012.

[19] A. Broder et al., "Min-wise independent permutations", Proc. Theory of computing, ACM Symposium, New York, USA, pp. 327-336, 1998.

[20] A.Broder, “On the Resemblance and Containment of Documents”, Proc. Compression and Complexity of Sequences, Washington, DC, USA, pp. 21-29, 1997.

[21] R. Panigrahy, “Entropy-based nearest neighbor algorithm in high dimensions”, Proc. ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, USA, pp. 1186-1195, 2006.

[22] A. Dasgupta, R. Kumar, and T. Sarlós, “Fast Locality-Sensitive Hashing”, ACM conference, New York, USA, pp. 1073-1081, 2011.

[23] Y. Hua et al., "Locality-Sensitive Bloom Filter for Approximate Membership Query", IEEE Transactions on Computers, vol.61, no.6, pp. 817-830, June 2012.

[24] N. Jain, M. Dahlin, and R. Tewari, "Using Bloom Filters to Refine Web Search Results", Eighth International Workshop on the Web and Databases, Baltimore, Maryland, 2005.

[25] W. Ong et al.,"A parallel bloom filter string searching algorithm on a many-core processor", IEEE Conference on Open Systems (ICOS), pp.1-6, 2-4 Dec. 2013.

[26] K. Mi Lee, K. Myung Lee, "Similar pair identification using locality-sensitive hashing technique", Joint 6th International Conference on Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS), pp.2117-2119, 20-24 Nov. 2012.

[27] D. Gorisse, M. Cord, and F. Precioso, "Locality-Sensitive Hashing for Chi2 Distance", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.34, no.2, pp.402-409, Feb. 2012

[28] K. Ling; G. Wu, "Frequency Based Locality Sensitive Hashing", International Conference on Multimedia Technology (ICMT), pp.4929-4932, 26-28 July 2011.

[29] M. S. Charikar, "Similarity estimation techniques from rounding algorithms", In Proc. of 34 annual ACM symposiums on Theory of computing (STOC '02). ACM, New York, NY, USA, pp. 380-388.

[30] M. Slaney, Y. Lifshits and Junfeng, "Optimal Parameters for Locality-Sensitive Hashing", Proc. of the IEEE, vol.100, no.9, pp.2604-2623, Sept. 2012.

[31] A. Stupar, S. Michel and R. Schenkel, "RankReduce - processing K-Nearest Neighbor queries on top of MapReduce", In LSDS-IR,2010.

[32] D. Knuth, "The Art of Computer Programming", vol. 3, 1973.

[33] P. Ciaccia and M. Patella, "Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces", In ICDE, pages 244–255, 2000.

[34] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions", Comm. ACM, pp. 117-122, 2008.

[35] B.H. Bloom, “Space/time tradeoffs in hash coding with allowable errors”, Comm. ACM, vol. 13, no. 7, pp. 422–426, July 1970.

[36] S.K. Pal, P. Sardana And K. Yadav, "Efficient multilingual keyword search using bloom filter for cloud computing applications", Fourth International Conference on Advanced Computing (ICoAC), vol. 1, no. 7, pp. 13-15, 2012.

[37] “Bloom Filter” [online] available at http://bytescrolls.blogspot.in/2011/05/til-bloom-filters.html, last accessed on 30th April, 2014.

[38] L. Qin et al., “Multi-probe LSH: Efficient indexing for high-dimensional similarity search”, Proc. VLDB, 2007.

[39] “LSH and General hashing” [online] available at http://cybertron.cg.tu-berlin.de/pdci08/imageflight/nn_search.html, last accessed on 30th April, 2014.

List of Publications

- S.S. Chauhan and S. Batra, "Finding Similar Items using Locality Sensitive Hashing and Bloom Filter", 2014 IEEE International Conference on Advanced Communication Control and Computing Technologies (ICACCCT-2014), ISBN No. 978-1-4799-3914-5/14.[Accepted]