Locality Sensitive Indexing for Efficient High-Dimensional Query Answering in the Presence of Excluded Regions

by

Aneesha Bhat

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved December 2015 by the
Graduate Supervisory Committee:

Kasim Selçuk Candan, Chair
Hasan Davulcu
Maria Luisa Sapino
Mohamed Sarwat

ARIZONA STATE UNIVERSITY

May 2016

ABSTRACT

Similarity search in high-dimensional spaces is popular for applications like image processing, time series, and genome data. In higher dimensions, the phenomenon of curse of dimensionality kills the effectiveness of most of the index structures, giving way to approximate methods like Locality Sensitive Hashing (LSH), to answer similarity searches. In addition to range searches and k-nearest neighbor searches, there is a need to answer negative queries formed by excluded regions, in high-dimensional data. Though there have been a slew of variants of LSH to improve efficiency, reduce storage, and provide better accuracies, none of the techniques are capable of answering queries in the presence of excluded regions.

This thesis provides a novel approach to handle such negative queries. This is achieved by creating a prefix based hierarchical index structure. First, the higher dimensional space is projected to a lower dimension space. Then, a one-dimensional ordering is developed, while retaining the hierarchical traits. The algorithm intelligently prunes the irrelevant candidates while answering queries in the presence of excluded regions. While naive LSH would need to filter out the negative query results from the main results, the new algorithm minimizes the need to fetch the redundant results in the first place. Experiment results show that this reduces post-processing cost thereby reducing the query processing time.

DEDICATION

To my family.

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my advisor, Dr. Kasim Selcuk Candan for his guidance throughout my research. He gave me the freedom to explore new ideas and approaches, and at the same time, corrected me when I deviated. Discussions with him would always clarify complex concepts. In addition, these discussions would provide pointers to new concepts and motivate us to come up with new ideas.

I would like to take this opportunity to thank Dr. Maria Luisa Sapino who has given me valuable feedback in all the research discussions we have had. I would like to thank my committee members, Dr. Hasan Davulcu and Dr. Mohamed Sarwat who have supported me and given me valuable inputs. I would to thank the Emitlab group members -Silvestro Roberto-Poccia, Jung Hyun Kim, Xinsheng Li, Xilun Chen, Shengyu Wang, Sicong Liu and Yash Garg. It was great to have intellectual and non-research discussions with you all. Thanks to Parth Nagarkar for his guidance from the very beginning. The countless brainstorming sessions have helped me advance in my thesis quickly.

I am grateful to Arizona State University and the School of Computing, Informatics, and Decision Systems Engineering for providing me with the infrastructure and resources during my graduate studies. I would like to thank Christina Sebring, who has advised me at every step regarding the procedures and deadlines.

I am extremely thankful to Dr. Erik Johnston, Dr. Dara Wald and Dr. Ajay Vinze for giving me the opportunity to work as a Graduate Research Assistant in their projects. These projects have given me opportunities to apply my computer science skills in sustainability.

I am thankful to my husband Adithya Murthy for motivating me to pursue my

dream of doing research. His never-ending patience and encouragement has led to my success. I am grateful to his mother Radha Murthy and father, U. Vishnumurthy for supporting me and my decisions. I am thankful to my mother, U. Vijayalakshmi, father, Manohara P and brother Manoj Bhat, for everything. I am also thankful to all my friends and everyone else who have helped me reach where I am today.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Growing use of technology is resulting in vast amounts of multimedia data in different forms like audio, video, and images. Many of the multimedia applications are represented as *high-dimensional data*[18]. High dimensional data is the representation of an object using a group of *features*(which define the object) called a *feature vector*. Color, texture, and shape descriptors are some of the examples of features that describe a multimedia object[49]. Due to the amount of information stored in each high-dimensional data object, their sizes are generally high. Efficient storage and retrieval of high-dimensional data is crucial for everyday applications. With enhanced mechanisms for storage, there is always a need for efficient retrieval techniques to get the required information.

Depending on the context and type of the application, there are various search techniques employed in multimedia applications. Some of the frequently used retrieval techniques which fall into the category of *similarity searches* in high-dimensional spaces are the following [49]:

- *Exact Searches* : To find objects identical to the given query.
- *Range Searches* : To find all objects in the data space which are within a given distance (*range*) from a query feature vector as shown in Figure 1.1(a).
- *Nearest Neighbor Searches* : To find an object which is the most similar to the query feature vector. This search can be generalized to find k nearest objects to the feature vector. Figure 1.1(b) shows a k-nn search where, for $k = 3$, the results returned are the points a, b and c.

- *All-closest-pairs-queries* : To find all possible pairs of objects which are similar to each other in the data space.

The formal definitions for Range query and Nearest Neighbor query are as follows:

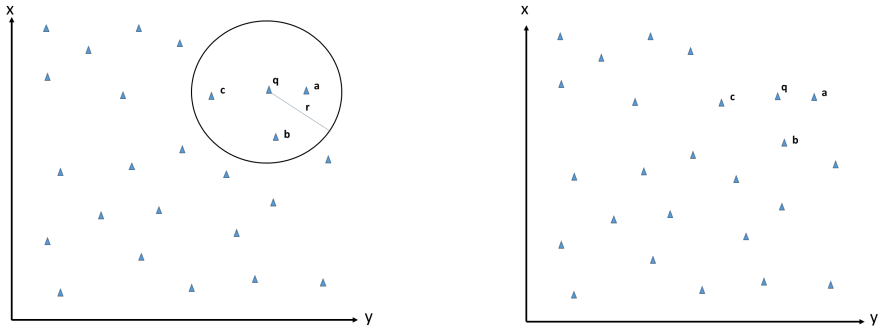
Definition 1.1. ***Range Query****, In the given space D, let q be the query point and $dist(v_1, v_2)$ be a function which computes the distance between any two points v_1 and v_2. Range Query retrieves all the points in the data space within a distance of r, where $r > 0$.*

$$RangeQuery(q, r)_{dist} = \{v \epsilon D | dist(q, v) \leq r\} \tag{1.1}$$

Definition 1.2. ***k-Nearest Neighbor Query****, In the given space D, let q be the query point and let $dist(v_1, v_2)$ be a function which computes the distance between any two points v_1 and v_2. K-nearest neighbor Query returns k closest points in the data space, where $k > 0$.*

$$k - nn\ Query(q, k)_{dist} = \{R \subseteq D | \forall v_r \in R, v'_r \notin R, dist(q, v_r) \leq dist(q, v'_r) \wedge |R| = k\} \tag{1.2}$$

Similarity searches in high-dimensional data is very popular for various applications involving image, genome, documents, and time series data [28]. There are various indexing mechanisms to solve the problem of nearest neighbor searches and range searches. Some of the well-known techniques include Quadtree [22], R-Tree [30], R+-Tree[50], R*-Tree [9], KD-Tree [10], SR-Tree [34], and TV Tree [36]. Due to the curse of dimensionality, these indexing techniques are rendered inefficient when the number of dimensions increase over ten[58].



(a) Range Query (b) K-Nearest Neighbor Query

Figure 1.1: Similarity Searches In 2-d Spaces

Dimensionality Curse

As the number of dimensions increase, most of these index techniques are rendered inefficient. As the dimensionality increases, distance between any two given points tends to be similar. Near neighbor loses its meaning and also makes similarity searches more complex due to the sparsity of data space. High dimensional data contains a lot of dead spaces and the conventional space partitioning techniques split the data irrespective of presence of data points. This results in a lot of redundant information being maintained in the index structure. Due to this, it may so happen that a query processing would result in an empty data page pointing to the space [12]. Index structures like R-Tree using Bounding Regions would result in a lot of overlapping Minimum Bounding Regions (MBR) which again results in an expensive query processing. The problem tree structures encounter is that when, dimensionality increases, almost all the nodes in the tree are checked for similarity [58].This leads to a space complexity of $O(n)$ or sometimes worse[51]. A simple sequential search over the dataset beats range searches using above methods.

However, a wide range of applications require similarity search in high-dimensions

like text or document search, audio and video retrieval [19], data compression [27], and pattern recognition [16]. This has given rise to the use of approximation techniques which provide faster results. Approximate near neighbors results are proven to provide similar results as sequential searches but in lesser time [32]. The gain in time with these approaches compensate for the occasional misses in the results.

Definition 1.3. ***c-approximate R-near neighbor, (r,c)-NN*** *[5] Given a data set D in d-dimensional space, a query point q, error probability δ and range r, if there is a point v, where $v \epsilon D$, such that $||q, v|| \leq r$, then $(r, c) - NN$ returns a point within distance cr with a probability of $1 - \delta$.*

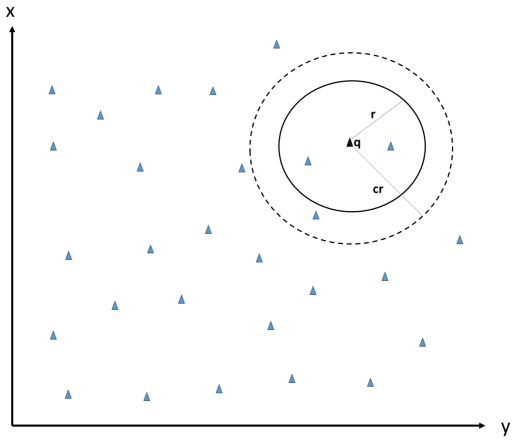


Figure 1.2: rc-Near Neighbor Query

Definition 1.4. ***Approximate k-nearest neighbor*** *[37] Given a data set D in d dimensional space, an integer k, and a query point q, k-NN query needs to return k points $v_1, ..., v_k$ in such a way that $||q, v_i||$ is at most $(1 + \epsilon)$ times more than the $||q, v_c||$, where ϵ is a small real number and $p_c \subset D$ and p_c is the closest point to q in D.*

Locality Sensitive Hashing (LSH) [28, 32] has been proven to be an efficient indexing mechanism to answer range queries and nearest neighbor queries in high-

dimensional databases. This technique makes use of hash functions that project the points in the original space to a lower dimensional space. It is based on the premise that points closer in the original space hash to the same buckets in the projected space with higher probability. Points hashed to the same bucket as the query are processed to fetch the results. LSH is used in a range of applications that deal with unstructured data like similarity search in tweets [53].

Queries With Excluded Regions

In addition to the similarity searches discussed above, there is a need for answering negative queries efficiently. A large number of applications have queries which have a main region of interest and a negative query indicating the region which needs to be excluded. In the case of multimedia objects, there are searches to find objects similar to a particular query feature f_1 and at the same time, not similar to a query feature f_2. This gives rise to a scenario where the query results comprise of a range query with f_1 as a query point and without the results of the range query with f_2 as query point. Here, the range query with query point as f_1 forms the *main query* or the main region of interest and the range query with query point as f_2 forms a *negative query*.

To further illustrate the above use case, the following are applications where this technique would be useful.

Document Search

A documents collection like web pages are often queried for similarity searches. Due to the size of the document collection, efficient indexing mechanisms are required to process the query quickly. Query file and documents are both considered to be a bag of words and searching using vector similarity is an easy technique to find results.

However, the similarity between documents only leverages the word(in terms of n-grams of characters) similarity. It does not make use of the semantics of the words. There are plenty of words in English which have different meanings depending on the context. If we know that the user is not interested in a specific subset of queries due to relevance feedback, we can show the most relevant results to the user by removing the uninteresting results. Consider the main query Q to be the search string or search file, which is a generic query that might mean different things in different contexts. Consider a query Q', such that $Q' \subset Q$ and the user is not interested in results returned by Q'. Here, Q forms a main query and Q' forms an excluded region or a negative query.

Image Retrieval

Image data can be represented as a set of features in space, where the number of features could range from a few tens to thousands. A feature in image data could be color, shape, contours, saturation, luminance or textures. There has been research in the area of application of LSH for image databases. To illustrate our problem, consider a collection of images in an image database consisting of millions of images. We have a query image Q_i, which displays an image of a street. We would like to retrieve all the images from the database that resemble this query image. Concurrently, we would want to exclude all the night time images of the street that are dark. Another query Q_n forms the negative query, the results of which we would like to exclude. While linear search on the whole database is deemed to be expensive, approximate methods can be used to fetch the results satisfying closeness to Q_i and exclusion of results satisfying the negative query Q_n.

Retrieving Music

Approximate nearest neighbor queries are very practicable in music search. Audio data can be best represented as high-dimensional data and each track can be represented as a set of features. These features are matched during a similarity query. [15, 14] discuss feature extraction in audio, similarity measure among audio shingles and finding closest music track using Locality Sensitive Hashing. Let Q_a be a query audio track and we would like to find all other audio tracks which sound similar. In addition, we are interested in finding those audio tracks which are instrumental. Given a vocal track, Q_v, we would want to perform a negative query using Q_v on the results returned by similarity search on a Q_a. Here, Q_a can be considered as a main query and Q_v as a query for the exclusion whose results we want to eliminate.

To elucidate further, the problem this thesis is trying to solve is described in the following section.

1.1 Problem Statement

The main goal of this research is to be able to query a region of interest while avoiding smaller region(s) within, which needs to be excluded. Figure 1.3 illustrates a simple example query : Let us assume we have a query q with radius r, which we are interested in and a query q' with radius r' which we are not interested in. The smaller region formed by query q' forms an *excluded region* in the main region of interest. If the region covered by q is Reg and region covered by q' is Reg', then $Reg' \subseteq Reg$. This thesis tries to efficiently answer such queries by minimizing the fetching of candidate points that fall within the excluded regions. The problem can be generalized to cases where $Reg \cap Reg' \neq \varnothing$.

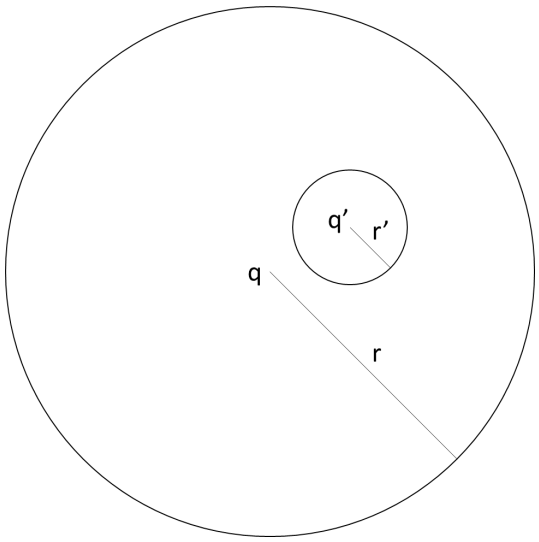


Figure 1.3: Query Region Of Interest,q, With an Excluded Region,q'

1.2 Thesis Contributions to Research

A naive approach to solve the problem is to perform a single query q, using LSH, and then compute the distance from the query point to all the candidate objects to remove the candidates which fall into the excluded region. This will give us only the points which lie within the region of interest. But this approach is inefficient since we need to retrieve and filter redundant candidates. This research provides a solution that avoids fetching the candidates that fall within the excluded region.

- Hash functions of LSH are used to project the original space of d dimensions into a lower dimension space, k (number of hash functions in a hash table).
- Z-ordering is applied on the reduced k dimensional space to generate a one-dimensional ordering of the projected points. The reduced space is stored in a data structure which utilizes the hierarchical nature of points for query processing.
- Range queries are performed intelligently on this data structure to achieve the

required results of positive and negative queries.

Given the range of applications like music retrieval, text, document search, and image retrieval, handling negative queries in an intuitive manner will be useful in many scientific and analytical multimedia applications. The algorithm provides a variation of Locality Sensitive Hashing method to provide a space efficient, time efficient data structure with quality guarantee of results.

- Firstly, this reduces a huge amount of processing cost required to filter out the unwanted data points from the candidate results.
- Secondly, we build the LSH index on a hierarchical structure which can answer range queries for multiple radii. The index is constructed for the smallest possible radius for the dataset, but stored in a manner using which larger radii range queries can be answered. This space efficient index structure is in the order of $O(n)$ which uses minimal memory.

1.3 Organization of the Thesis

The organization of the rest of the thesis is as follows:

- Chapter 2 provides a literature survey on the popular indexing techniques in high-dimensional database
- Chapter 3 introduces background concepts about Locality Sensitive Hashing and Z-order curves.
- Chapter 4 presents the proposed enhancement over the existing naive LSH.
- Chapter 5 discusses the experimental setup, results, and analysis of the experiments.

- Chapter 6 concludes the thesis and discusses future work.

Chapter 2

RELATED WORK

Over the years, a large number of indexing techniques have been developed for indexing in high-dimensional databases. Various techniques include one-dimensional ordering, space partition techniques, data partitioning methods and approximation methods.

2.1 Space Filling Curves

A simple yet interesting approach is to provide one-dimensional ordering to high-dimensional data. This can be achieved by *space filling curves*[6]. These space filling curves span across the data space covering all the data points, to transform the high-dimensional space into a one-dimensional sequence. There are simple space filling curves like the column order or the row order and complex ones making use of concepts of bit interleaving [55], quad code[22] and location code [3]. Some of the popular space filling curves are Gray Code [20, 21], z-order curve[45, 40] and Hilbert's curve [33]. Row ordering or column ordering are called concatenation methods since the code corresponding to the data point is derived by concatenating the *x-coordinate* value with the *y-coordinate* value(row or column ordering depending on whether the x-coordinate is most significant or y-coordinate) [49]. On the other hand, z-curve is a bit-interleaving code. In this method, the code representing the object is formed by alternating the bits of the x-coordinate and the y-coordinate in two-dimensional space. Bit concatenation leads to long narrow searches whereas bit interleaving methods like z-curve facilitates compact square searches. Hilbert curves and z-order curves exhibit an interesting behavior called the *fractal* structure. They are self-similar and tend

to repeat the same pattern at multiple scales. This beautiful behavior proves to be useful to perform range queries and partial queries at different levels of granularities. Compactness and the fractal nature of the curves makes Hilbert or z-curve more popular than the row-ordering or column ordering curves.

2.2 Multi-Dimensional Index Structures

One of the basic yet highly prominent foundational construct is the hierarchical subdivision of data. Common examples are the B-Trees or the B+-Trees [8]. The key idea of such index structures is that they provide sorting of data. This sorting can prove useful while data is stored on disk. If data points are close in the original space, sorting ensures that they are close on disk.[13]. These methods can be classified into *space partitioning* techniques and *data paritioning* techniques [23]. In space partitioning techniques, the space is divided into hyperplanes irrespective of the distribution of data. This forms a disjoint set of regions with data. In data partitioning techniques, the space is divided based on the distribution of the data. This may result in overlapping regions of data.

Grid files is one of the older indexing techniques in which the space is divided using a grid like structure. Each cell within the grid contains data points which can be accommodated within a page while storing[43]. The disadvantage of such a structure is that if there are less points, then the space in the page can go wasted and if there are more points, then the points can span across more than one page.

Quadtrees, although similar to grid files, are an improved index structure where the space is partitioned taking into consideration the data points [22]. The root of the quadtree covers the entire space. Each internal node has four children in a two-dimensional quad tree. Each incoming point subdivides the two-dimensional space into four partitions which forms its children subtree[4]. Using this approach a

hierarchical structure is built. A node is split when the number of points increase over a predefined limit. When the same principle of indexing is applied to two-dimensional data, it is referred to as Octree. This approach can be generalized to d-dimensions.

Another popular index structure for high-dimension data space is the kd-tree which supports multiple dimensions. It forms a binary search tree structure, where each internal node divides the space into two along a single dimension. The space is split along different dimensions at different levels of the tree in a round-robin fashion [13].

R-Tree is a another popular multi-dimensional index technique which uses minimum bounding regions (MBR) to store the objects. R-Tree is a generalized version of B+-tree which handles multiple dimensions along with offering efficient space utilization. MBRs are constructed such that they form a minimum bounding region for the point object. To form a hierarchical tree structure nearby MBRs are grouped to form a larger MBR. Though R-Tree is one of the very popular techniques that is employed in a large number of spatial and other commercial databases, it does have a drawback due to the overlapping MBRs. R+-Tree and R*-Trees provide enhancements over the naive R-Tree.

To overcome the problem of curse of dimensionality seen in these tree based index structures, there has been extensive research on approximate indexing techniques. Vector Approximation Files and Locality Sensitive Hashing are popular approximation indexing methods.

2.3 Vector Approximation Files (VA-Files)

In [58], the authors establish that the popular partitioning methods provide a linear complexity which is, most of the times, surpassed by the naive sequential scan. VA-File is a method which optimizes linear scan for performing similarity searches.

Using VA-File, each data object is represented with the help of (a) an approximation cell represented by b(user given parameter) bits per dimension and (b) a pointer to the list of data objects on the actual disk. The data space is divided into 2^b cells and each data object is represented by one of these cells [11]. Every cell is represented by a bit string formed by the concatenation of bit values of ranges in each dimension. Hence a VA-File index is essentially an array of these bit-string values which represent the grid cells and each cell contains a pointer to the data objects on disk that fall into the cell. Given a range query, a linear scan of the vector approximation cells is performed. Every cell is checked to see if it is a candidate cell that might contain objects that fall under the range. This is estimated by computing the Euclidean distance from the query point to the boundaries of the VA cell. On finding a potential cell, Euclidean distance with all the objects within the cell is computed, to filter out only relevant results. [11] describes a *Simple Search Algorithm* and *Near Optimal Algorithm* to perform similarity searches.

2.4 Locality Sensitive Hashing and its Variants

The basic LSH was initially proposed by Indyk and Motwani in [32]. In this work, the authors explain LSH taking into consideration the Hamming distance. In [28], the authors further improve the basic LSH described earlier in [32], by optimizing the previous approach to provide faster query time. In addition, the authors generalize the previous approach into external memory. It was in [17], that LSH was first shown to be designed for Euclidean distance. In this paper, the authors described the well known technique of projecting the d-dimensional vector on a random vector and then quantizing it into predefined buckets of width w to hash a data point. Data points that fell into same buckets as the query point were chosen as candidate objects. k such hash functions needed to be satisfied to be chosen as a candidate. This reduced

the number of false positives. More (l) hashtables were constructed to reduce the misses.

An improvement over the original method called LSH Forest was proposed in [7]. The authors suggest a method to fetch the similar points without having to depend on data to tune LSH index. The authors show that this improved LSH's performance for skewed datasets. Each layer is represented by a prefix tree data structure. The paths of the prefix trees from root to leaves represent the k hash values in the layer. Querying is a two-phased approach. The first phase which is a top-down approach, finds the path to the leaf that contains the largest prefix of hash codes common to the hash code formed by the query point. In the second bottom up phase, the nodes are traversed upwards fetching candidates. If the lowest level points to all the required candidates, the bottom-up traversal stops there, else it continues until required candidates are fetched. A collection of such trees represent different layers of a LSH, to form the LSH Forest.

In [54], the authors construct a compound hash key of length k. This represents the data object in a reduced dimensionality of k. A z-order code for this vector in a low dimension is computed. These values are stored in a B+-Tree, that further transforms the points to a one-dimensional space. Given a query, the largest prefix of the z-code of compound hash keys of the query is found in the B+-Tree. This is the node of the B+-Tree preferred to fetch the candidates. In case more candidates are required, the previous and next nodes of the tree are probed. A collection of such trees forms an lsb-forest.

Another approach called Entropy based LSH [57] overcomes the space requirement compared to previous methods. This method assumes that a point close to the query is known and those buckets are probed that are calculated to have higher success probability depending on the point. Using the underlying idea of entropy based LSH,

Multi-probe LSH probes nearby buckets for candidate results [39]. In this approach, rather than creating multiple hash functions, very few hash functions are created. If more results are required, nearby buckets are probed to fetch results. A work on query range sensitive LSH provides an algorithm that generates a probe sequence based on query range rather than a fixed probe [29]. There have been enhancements to LSH to handle dynamic data sets and when queries are known [44, 31].

In [24], the authors publish a new variant of LSH based on counting collisions in hash buckets. Instead of the traditional static L compound hash functions each consisting of k hash functions, this work suggests creating m compound hash functions. A candidate data vector is chosen if its hash code collides with a query point's hash code in at least k of the hash functions. In case no data point collides in k hash functions, the paper employs a virtual re-ranking scheme to expand the width of the hash bucket thereby increasing the chance of collision.

All of the above methods suffer from a major limitation that many random Input/Output (I/O) operations are required to fetch the candidate results. Sorting-Keys LSH (SK-LSH) overcomes this drawback by storing the data points on disk such that nearby points are stored closer to each other than the farther points [38]. To achieve this, the authors define a new distance measure and sorting mechanism for data points based on the prefix in the compound hash keys, and the data points are stored in sorted order. While searching for the candidates, they ensure that minimum number of disk pages are accessed. [56] gives a detailed explanation of various hashing techniques used for similarity searches.

Recent research in the area of similarity searches in high-dimensional data, depends on the distribution of data. One such technique is Data Sensitive Hashing [26] which makes use of adaptive boosting to generate hash functions. Another technique called Data-Dependent Hashing [59] works in two levels. In the first level, data is

divided into clusters of similar distribution. Hash table indexes are built for each of these clusters. For a given query, clusters with high probability of finding result are chosen and queried using the index structure of that cluster.

A very recent index mechanism based on LSH is called Selective Hashing [25]. The authors create multiple indices with different radii. Each object is a part of only one of the indices creating a disjoint set of indices. During the query, all the indices are looked into and relevant candidates are chosen. This method proves to be space efficient and provides results with same recall as previous methods.

In [41], the goal of the authors is to efficiently solve a range query workload in a one-dimensional space. In order to do this, they create novel query plans for answering range queries. In this work, the authors introduce an *Exclusive* query plan for answering excluded regions. Similarly, in [42], the authors also use *Exclusive* query plans to efficiently solving range query workloads for two-dimensional spaces. These work leverage the hierarchical nature of the data to provide efficient query processing plans. They create cost models and query plans to answer a range query workload for one-dimensional and two-dimensional spaces. The goal of my work is to answer queries with excluded regions in high-dimensional spaces.

In spite of having multiple indexing techniques, none of these techniques directly intend to solve the problem of answering query with excluded regions. To answer range queries comprising of excluded regions or negative queries, the main query is executed using one of these techniques and the irrelevant results need to be eliminated. Hence, there is a need for an effective algorithm that can handle negative queries nevertheless with similar guarantee of results.

Chapter 3

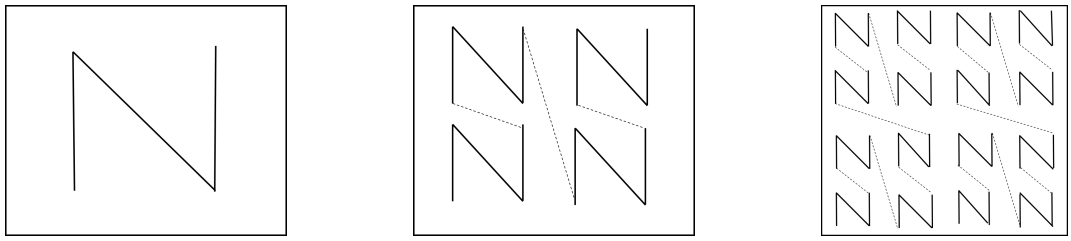
INDEXING IN HIGH-DIMENSIONAL SPACES

This section provides background information and concepts based on which the proposed algorithm is designed. Firstly, we discuss the z-order curve [45, 40], computing z-codes and its hierarchical property. Next, we discuss the Locality Sensitive Hashing [32] and its quality guarantee.

3.1 Z-order Curve

Z-order or Morton order curve is a space filling curve that provides a one-dimensional ordering to two or more dimensional data. At the same time, these one-dimensional ordering preserve the locality of the points in the original space. Due to its simplicity in mapping to one-dimensional ordering and its ability to preserve locality efficiently, Z-order curve has been used as a popular technique[6].

Z-curve falls into the category of space filling curves that possess *fractal* property. A fractal is a structure that exhibits a pattern of self-similarity in different resolutions. Figure 3.1 shows the multiple scales of the z-ordering curve. In Figure 3.1(a), we see that the whole space is covered by one single z. The z curve can be seen as covering 4 different regions in space - beginning from *South West*, next *North West*, continuing to *South East* and finally *North East*. In Figure 3.1(b), we see a zoomed in version of the previous z-curve, with each of the four regions having their own z-curve. And finally in the Figure 3.1(c), the z-curves cover regions with finer granularity.



(a) 1st Order (b) 2nd Order (c) 3rd Order

Figure 3.1: Z-order Exhibiting Fractal Property [35]

3.1.1 Computing the Z-codes

Bit concatenation and bit-interleaving are two simple methods to map the multi-dimensional point into one-dimensional ordering. Z-curve makes use of the bit-interleaving method.

Figure 3.2 illustrates the mapping of a point to the corresponding z-curve value. Consider the region highlighted in red, in the figure. If we were to number each cell as the z-order curve sweeps through the region, starting from the bottom left cell as 0 and the top right cell as 15, the red region would be numbered as 14. Though it appears straight forward to count the cells as the z-curve sweeps, consider high-dimension space with more than twenty dimensions and more partitions in space. Nevertheless, z-curve wins, thanks to the easy mapping technique. Notice that the red region is formed by $x = 3$ and $y = 2$. The z-code of the cell is formed, by interleaving the digits in the binary representation of the values of the two co-ordinates. The right side of the figure shows the bit-interleaving of 11 and 10 to arrive at 1110 which is 14.

To formalize the mapping, let us consider a two-dimensional space(for the sake of simplicity) with *x-axis* and *y-axis* being the two dimensions. Both the co-ordinates are divided into k equal regions. To simplify the geometry, let us assume that k is a

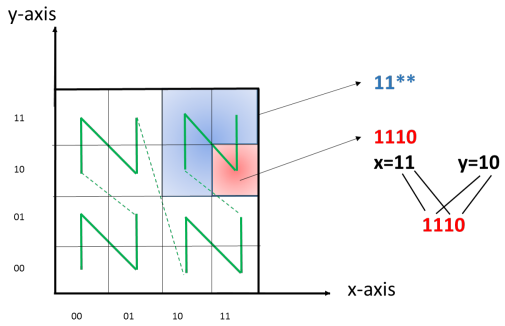


Figure 3.2: Mapping a Point to Z-code

power of 2. Each of these divisions can be represented by $log_2 k$ binary digits. Let a point $A = (x, y)$ be represented as $A = (x_{k-1}, ..., x_0), (y_{k-1}, ..., y_0)$ [49]. Z-order code for the point A is given by the following

$$zcode(A) = x_{k-1}y_{k-1}...x_1y_1x_0y_0 \tag{3.1}$$

The simple mapping technique has made the mapping from high-dimensional spaces to one-dimensional ordering very efficient. From the Figure 3.1(c), it is clear that the ordering is very compact. This implies that, as the granularity of the curve formation increases, z-curve fills smaller regions at a time and grows to cover the whole space while retaining the z nature [35].

3.1.2 Hierarchical Nature of the Z-order Curves and Range Queries

Locality preserving nature of z-curves have proven to be useful in range queries [46]. Using a one-dimensional index structure like the B+-Trees, one can map the points on high-dimensional space to one-dimensional ordering in a B+-Tree. However, these mappings do not capture the locality in its entirety. There are cells which appear next to each other in the Figure 3.2, but their z-code numbers have a lot of other cells in between them.

Figure 3.3 shows the hierarchical representation of the z-orderings of the cells in

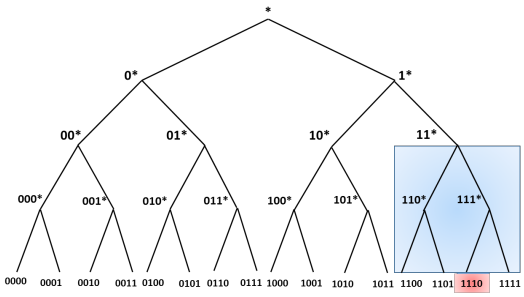


Figure 3.3: Hierarchical Representation of Z-curve Ordering

3.2. Let us consider a query ranging in the blue shaded box. This region covers one complete z square of the $3rd$ order. It is interesting to note that due to the hierarchical nature of the z-curve, the blue box can be represented as $11**$, where $*$ is a *don't care* [13]. Any of the four cells represented by 1100, 1101, 1110 and 1111 can be represented by $11**$. Using this trait, a prefix based index structure can be constructed to answer range queries. For range queries spanning across different z regions, the query can be split accordingly into sub queries, to answer the range query efficiently. The proposed algorithm make use of this hierarchical nature of the z-curve to eliminate exclusion regions in the query.

3.2 Locality Sensitive Hashing

As discussed earlier, the need for faster processing of similarity searches in high-dimensional data has given rise to approximate indexing techniques, Locality Sensitive Hashing being one of the most prominent and efficient ones. This technique is based on the premise that points closer in the original space hash to same buckets. Points hashed to the same bucket as the query are potential candidates for results. A post processing step on the candidates gives us the final results. Over the years, many hash families have been devised to work with various distance measures.

Definition 3.1. ***Locality Sensitive Hashing*** *is formally defined as the following*

in R^d space. A family of hash functions H is considered to be locality sensitive in (r, cr, P_1, P_2), if the following condition is satisfied for any two points q and v in R^d. [17]

1. *if* $||q, v|| <= r$, *then* $Pr[h(q) = h(v)] \geq P_1$
2. *if* $||q, v|| >= cr$, *then* $Pr[h(q) = h(v)] \leq P_2$

The Definition 3.1 implies that, if q and v are within the required distance of r, then the probability of both the points hashing to the same bucket is very high, i.e., greater than P_1. If the distance between q and v is greater than cr, then there is a very low probability (less than P_2) that both the points hash to the same bucket. The same is illustrated using Figure 3.4. It is crucial to maintain $P_1 > P_2$ to achieve the required quality guarantee. The gap between P_1 and P_2 is amplified by using multiple hash functions.

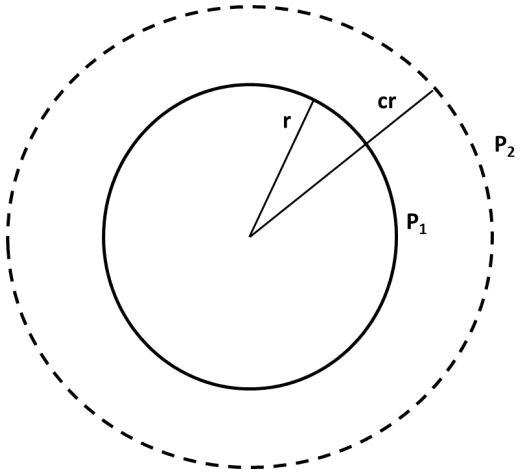


Figure 3.4: Range Search With Probability P_1 And P_2 for Radii r and cr

3.2.1 Hash Function

The quality guarantee of Locality Sensitive Hashing technique depends on the effectiveness of hash function chosen in the Definition 3.1. It is interesting to note that, in several applications, hashing is used as a technique to uniquely identify a given object. The motive in the case of such hash functions is to hash the objects into as different buckets as possible. Unlike such hash functions, LSH employs hash functions which have the ability to hash similar objects into same buckets [48]. This is the key idea behind use of hashing in similarity searches. Different hash families have been discovered for different distance measures. For instance, when the distance measure used is Hamming distance, the hash function is the value of a randomly chosen co-ordinate of the binary vector. Let us assume the binary vectors are of the form $\{0,1\}^d$ and the distance between any two points $v1, v2$ is given by the number of indices where the values of the indices do not match. The hash function for such Hamming distance is given by $h(p) = p_i$ where $i \ \epsilon \ \{1, 2...d\}$ is chosen randomly[32]. Here, the probability P_1 is given by the number of co-ordinates in the vectors where the values match. If the radius is given by r, then $Pr(h(q) = h(v))$ is given by $1 - \frac{r}{d}$ (since the number of co-ordinates where the values can be different are at most r). Similarly, P_2 is given by $1 - \frac{cr}{d}$. It is clear that $P_1 > P_2$ holds, hence it is an ideal and simple hash function for Hamming distance family.

Since Euclidean distance measure is one of the most widely used distance measures, we assume Euclidean distance in all our further discussions. Let us consider a data set D containing n points. All points $v \ \epsilon \ D$ are of dimension d. Euclidean distance (denoted by $||v - q||_{l_2}$) between any two points in space is given by

$$||v - q||_{l_2} = \sqrt{\sum_{i=1}^{d} (v_i - q_i)^2} \tag{3.2}$$

In the case of Euclidean distance, [17] was the first to devise a hash function that guarantees closeness of points in hashed space if the points were close to each other in the original space. An ideal hash function for the Euclidean distance family in R^d is given by the following:

The point v is projected on a random vector $\vec{a}$, displaced by a random value b and quantized into hash bins of size w. The random vector is taken from a p-stable distribution, constructed by choosing the values of each of the co-ordinates from a *Gaussian distribution* of mean 0 and standard deviation 1, $N(0, 1)$. Such a random vector is called a Gaussian random vector. It is proven that projections of two close points using a Gaussian random vector remain close (explained in detail in Section 3.2.2).

The hash function for Euclidean family is defined as the following:

$$h(\vec{v})_{a,b} = \left\lfloor \frac{\vec{v}.\vec{a} + b}{w} \right\rfloor \tag{3.3}$$

where $\lfloor . \rfloor$ is a floor operation, $\vec{v} \, \epsilon \, D$ is a point in the d dimensional space D, w is the bucket size, a is a Gaussian random vector and $b \, \epsilon \, [0, w)$ is a random number. The geometric representation of the above equation is shown in the Figure 3.5. A is the scalar dot product of query vector q, and the Gaussian random vector $\vec{a}$. Similarly B is the scalar dot product of query vector v, and the Gaussian random vector $\vec{a}$. Quantizing the one-dimensional line into equal buckets of width w gives the hash bins into which the projections fall.

Using just one such hash function can lead to false positives and misses. In order to avoid false positives, many such hash functions are concatenated to form a Compound Hash Function. Hash values to which the points map to, are called the Compound Hash Key (CHK). Many such CHKs can be used to avoid the misses. Next section provides the theoretical guarantee of LSH using multiple hash functions and layers.

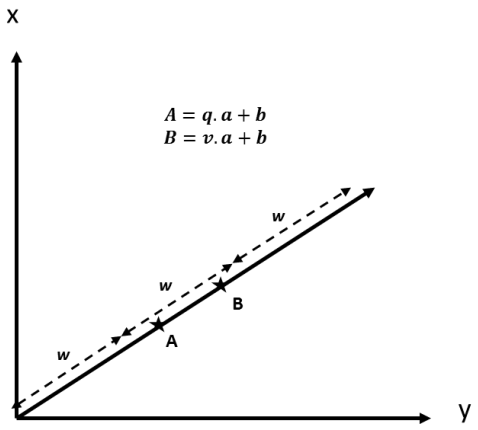


Figure 3.5: Geometric Representation of Hash Function for Euclidean Family

3.2.2 Quality Guarantee

This section discusses the probability guarantees for Euclidean hash function. From Definition 3.1, we know that for a hash function to provide required success probability, it is necessary to hold the inequality $P_1 > P_2$. Note that the Gaussian random vector is generated using p-stable distribution. Projections of points that are close, are also close to each other in the projected space when a Gaussian random vector is used. The formal definition of p-stable distribution is the following[52]:

Definition 3.2. ***p-stable distribution*** *A distribution S is called p-stable distribution, if, for any real numbers $v_1, ..., v_n$ and random variables $X_1, ..., X_n$, $\sum_i v_i X_i$ has same probability distribution as $(\sum_i |v_i|^{\frac{1}{s}}).X$, where X is a random variable drawn from S.*

Gaussian distribution being a 2-stable distribution, has the following property [60]:

$$pdf(\vec{v}.\vec{p}) = |v|N(0,1). \tag{3.4}$$

The above equation implies that the probability distribution function (pdf) of a scalar

dot product of vector $\vec{v}$ with a Gaussian random vector,$\vec{p}$ is the same as magnitude of the vector $\vec{v}$ times the pdf of Gaussian distribution. This implies that for any two points, q and v, if the $||q, v||$ is less, the points are closer and the difference in magnification of the pdf is small. Hence, there is a high probability that the two points will collide into the same hash function. For points far away, the probability of collision decreases.

To compute the probability P_1, that two close points hash to the same bucket, we denote $f_p(x)$ to be the probability density function for p-stable distribution. Let $r = ||v, q||$ be the Euclidean distance between v and q and w be the hash bucket size. If $\vec{a}$ is a Gaussian random vector, the probability distribution of $a.v - a.q$ is the same as $r.X$, where X is a random variable drawn from the p-stable distribution. The probability that the two points v and q collide is given by

$$Prob[h(v), h(q)] = \int_0^w \frac{1}{r} f_2(\frac{t}{r})(1 - \frac{t}{w})dt \tag{3.5}$$

for L_2 space [17].

From Equation 3.5, we see that the probability of collision depends on the distance between the two points. The probability monotonically decreases when the distance between the points increases. Since P_1 denotes the probability of closer points and P_2 denotes the probability of farther points, $P_1 > P_2$. Figure 3.6 shows the change in probability as for different values of r when w is fixed. When the radius has a value of r, probability that any point within r collides with the query q in the hash functions, is given by P_1. As the radius value increases, say to cr, then the probability that any point outside of cr hashes to the same bucket as q is given by P_2.

The gap between *high probability* P_1 and *low probability* P_2 can be amplified by using many hash functions instead of a single hash function. Let $g(v)$ be a function of concatenation of k hash functions $h(i)$ where $i \, \epsilon \, 1, ..., k$. Each hash table is constructed

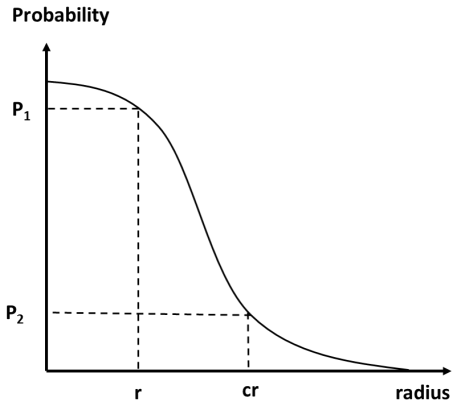


Figure 3.6: Range Search with Probability P_1 And P_2 for Radii r and cr

by using a family of hash functions G which reduces the dimension of the R^d space into a k-dimensional space. Let $G = \{g : R^d \mapsto U^k\}$, where $G(v) = h_1(v), h2(v), ..., h_k(v)$. The k hash functions are generated randomly and independently. A function $G(v)$ returns a concatenation of real numbers which forms a new hash key for the point v. Using a concatenation of k functions forms a stricter hash function than a single hash function $h(v)$ reducing the *false positives* that would have been returned otherwise. On the flip side, a stricter hash function $G(v)$ introduces *false negatives* in the results. To overcome this drawback, l layers of such hash functions are used. The index structure is constructed using l hash tables $G_1, ..., G_l$ and it is sufficient that the hash key of a point collides with the query point in any one of the hash tables. The parameters k and l can be tuned to achieve the required probability.

If P_1 is the probability that two points collide with each other, then the probability that two points collide with each other in all the k hash functions is given by P_1^k. It follows that the probability that two close points do not collide in at least one of the k hash functions is given by

$$1 - P_1^k.$$

Probability that two points do not collide in at least one hash function in all the l layers is given by

$$(1 - P_1^k)^l.$$

Hence, the probability that two points collide in all the k hash functions in at least one of the l layers is given by

$$SuccessProbability = 1 - (1 - P_1^k)^l \tag{3.6}$$

The parameters k and l are chosen to achieve the required probability guarantee.

Using these fundamental concepts, the next section discusses the enhanced algorithm for LSH for handling exclusion regions.

Chapter 4

ANSWERING QUERIES IN THE PRESENCE OF EXCLUDED REGIONS

LSH has been proven to give good results for approximate nearest neighbor searches and range searches in high-dimensional spaces. In this section, an enhancement is proposed on the existing LSH algorithm. This new approach is designed to handle range queries which are comprised of negative queries. To set the context, the relation between the radius of range query and the bucket size used in LSH is discussed first. The next section provides the design of the new algorithm. The design is first described for the hash function level, then built on for the compound hash key level and next for the whole index. Finally, the design and algorithms of query processing, are described.

4.1 Relation Between Query Radius, r and Bucket Width, w

A crucial step in the LSH procedure is to quantize the projection of the data object on to a random vector. The projection scale is divided into equally spaced buckets of width w (introduced earlier in section 3.2). Let v and q be two points in the data space D of dimension d. Let $r = ||v - q||$ be the Euclidean distance between the points v and q. Let a be the random vector of dimension d whose co-ordinates are drawn from the Gaussian distribution. The distance between the two points in the projected space is given by $a.v - a.q$. The probability that v and q collide with each other in the hashed space is given by the following [17]:

$$Prob[h(v), h(q)] = \int_0^w \frac{1}{r} f_2(\frac{t}{r})(1 - \frac{t}{w})dt$$

where, $f_2(x) = \frac{2}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$

Solving the integral gives us,

$$P_1 = 1 - 2norm(-\frac{w}{r}) - (1 - e^{-\frac{w^2}{2r^2}})\sqrt{\frac{2}{\pi}}\frac{r}{w} \tag{4.1}$$

where, for any random variable in a distribution with mean 0 and standard deviation 1, $norm(.)$ denotes its cumulative distribution function (cdf).

If $r^2 \geq w^2/2$, P_1 can be reduced to

$$P_1 \leq 1 - \sqrt{\frac{2}{\pi}}\frac{r}{w}(1 - \frac{1}{e}) \tag{4.2}$$

Given a probability P_1, it is evident from the Equation 4.2, that w is directly proportional to r. When the value of w is small, the bucket size formed by the intersection of k hash functions is small. Such a small bucket would hold less number of candidates (points closer to the query with high probability), hence answering smaller radii. Fetching more number of candidates can be accomplished by increasing the width of the bucket, w, within each hash function. This principle forms the basis of other popular LSH algorithms like Multi-Probe LSH [29]. In Multi-Probe LSH, when more number of candidates needs to be fetched, the nearby buckets are probed to fetch the candidates.

In [17], authors discuss the effect of changing w on ρ, where $\rho = \frac{\ln 1/p_1}{\ln 1/p_2}$. It has been experimentally shown by the authors that, after a certain value of w, ρ is not effected by increasing w further. However, a large value of w will further reduce the gap between P_1 and P_2. To overcome this problem, more hash functions in each layer is required, thereby increasing the query processing cost.

In the new algorithm, this principle is used to assign different bucket sizes, w and w' for the query and the excluded region respectively. This depends on a number of factors like the data distribution and the number of hash functions k. If r and r' are

two radii such that $r > r'$, and w and w' are the corresponding bucket widths, then $w > w'$.

4.2 Design of the Proposed LSH Index

In this section, the proposed hierarchical index structure using Locality Sensitive Hashing is explained which enables us to answer queries with excluded regions. As described earlier, LSH is used to perform dimensionality reduction on the data. Several data points can fall into the same hash code created by the hash function. Within each layer of the index structure, a hierarchical tree is constructed to depict the reduced space and the pointers to data points. This vector in the reduced space is converted to a *z-code*. We get a one-dimensional ordering for all the vectors in the reduced space. This is stored in a data structure which makes use of the hierarchy of the points to answer different range of queries. An algorithm is provided to retrieve the required range of queries while handling negative queries as well.

Figure 4.1 provides the overall workflow of the algorithm. The data in the original space is projected on a smaller dimension using the compound hash functions of LSH. This transforms the original data points to CHKs. The original space of d dimensions is reduced to the size of CHK. Z-codes are computed for each of the CHKs to get a one-dimensional ordering. The one-dimensional ordering is stored in a binary search tree to perform range queries. Range searches are performed on the search tree avoiding the range of excluded region to get the candidates at each layer. Candidates from all layers are combined and actual Euclidean distance is computed to filter out the false positives. The workflow is explained in detail in the rest of this chapter. The design is explained for a main query with one excluded region, however, the same design can be used to handle multiple exclusion regions.

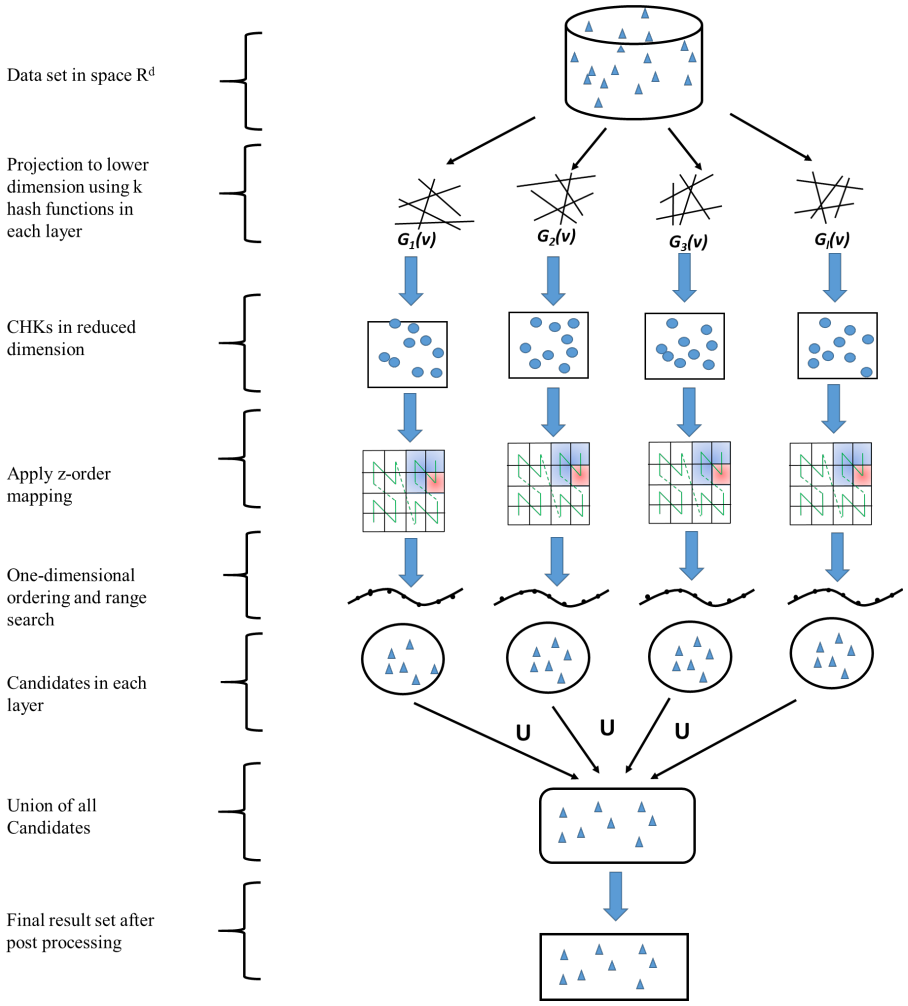


Figure 4.1: Workflow of Building Index and Query Processing

4.2.1 At a Hash Function Level

Consider the data space D in d dimensions consisting of n points. The first step is to convert the d dimensional point v, $(v \ \epsilon \ D)$ to a k-dimensional point. This is achieved by compound hash keys generated by using k hash functions in LSH, $G(o) = h_1(o), h_2(o), h_3(o)...h_k(o)$. For the Euclidean hash family, the hash function is formed by projecting the data vector v on a random vector a and by chopping the line into equally spaced buckets of width w as described in Equation 3.3. Let us

consider a query set which consists of a main query and a query for excluded region, of a smaller radius. In the previous section, we see that bucket size, w, is a function of the query radius, r. This demands an index structure which is sufficient to answer queries of both the radii. In the new approach, an implicit hierarchy is constructed to depict the various resolutions of the bucket size.

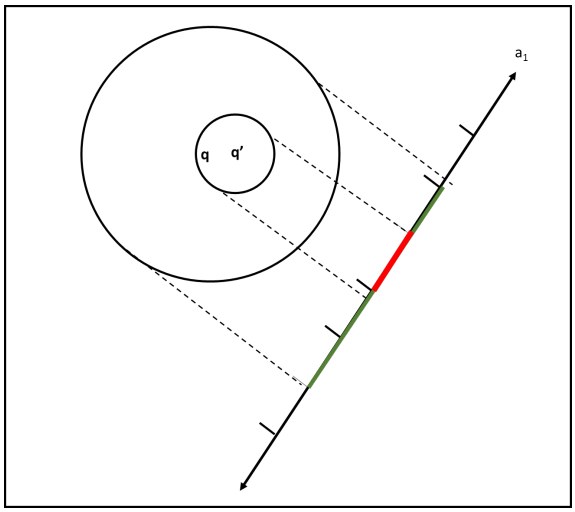


Figure 4.2: Projection of the Main Query and the Excluded Region on a One-Dimensional Real Line. The Projection of Negative Query Region is a Subset of the Projection of Main Query Region.

Figure 4.2 shows the projection of the main query q on the real line l in green. The projection of the negative query region is shown in red. We notice that the locality of the main query and the negative query is maintained in the projection. The goal is to fit the two different radii with optimal values of w (for the main query) and w' (for the negative query) such that the both w and w' divide the real line into 2^f and $2^{f'}$ buckets where f and f' are integers. This condition (though not necessary) simplifies the algorithm and the discussion in the next section when we map the hash values into a one-dimensional ordering.

For a given data set, we can fix w and w' to be the optimal bucket sizes, where r

and r' are the main query radius and negative query radius respectively. Hierarchical nature of the bucket size provides us with the flexibility to choose from possible options of w value whenever r is greater than r'. Quality of the results can be further tuned by varying the values of number of hash functions k. In the case of traditional LSH, the bucket size w is generally chosen to be significantly larger than the query radius r. Though this increases false positives, it does reduce misses. However, when the goal is to avoid the region in case of an exclusion queries, w needs to be as small as possible. A false positive in the excluded region results in a miss in the main region of interest. Such misses needs to be avoided.

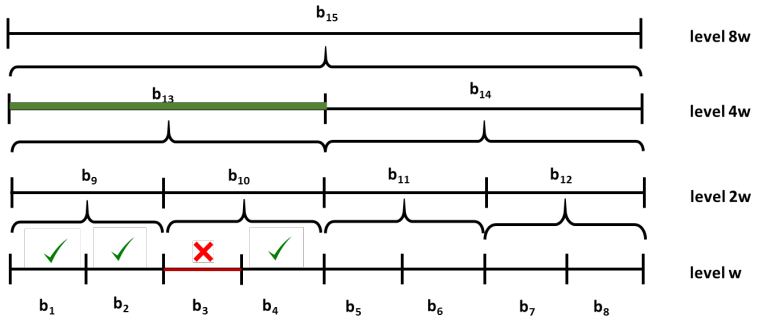


Figure 4.3: Hierarchical Bucket Structure (For Different Bucket Sizes) Of A Single Hash Function

A hierarchical representation of the bucket sizes is shown in Figure 4.3. To answer the query, q, we use a query at level $4w$ which falls into bucket (of a hash function) b_{13}. We use radius w to get the bucket (of hash function) which q' falls into. b_{13} comprises of b_1, b_2, b_3, and b_4 at leaf. The candidate results for this hash function are those points which are in the buckets (of hash function) b_1, b_2, and b_4.

$$b_{13} - b_3 = b_1 \cup b_2 \cup b_4 \tag{4.3}$$

In the case of a single hash function, it is clear that we can fetch the candidates in the buckets b_1, b_2, and b_4 without having to look into the bucket b_3. As in the case

of traditional LSH, we make use of multiple such hash functions in a layer, to build a compound hash key. Also, we use multiple layers to increase the accuracy of the results. The following sections discuss how to perform the intersection of buckets of different hash functions while maintaining the boundaries of the excluded region.

4.2.2 At a Hash Table Layer Level

A bucket at the level of a layer is formed by the intersection of k hash functions. For a point to be a candidate result of the query, it needs to collide with the hash values of the query point in all the k hash functions. In the problem of query set including negative queries, for each of the k hash functions, we have a larger range of candidates satisfying the main query and a smaller subset satisfying the negative query. The goal of this section is to devise a data structure such that we can efficiently fetch only required candidates. As a solution, we construct a tree index structure at each layer to represent the space projected using hash functions. The dimension of the new space is equal to the number of hash functions in a layer, k. The scale of each dimension is based on the number of buckets (within a hash function). The hash value of a data point, v, needs to collide with the hash value of the given query, q, for each dimension of the new space for v to be returned as a result.

As shown in Figure 4.4, q is the larger query and q' is a query with smaller radius. The region of interest in this figure is $q - q'$. Let us consider $k = 2$ hash functions in the layer, forming a Compound Hash Function (CHK), $G(v) = h_1(v).h_2(v)$. If the bucket width were to be set appropriately, and S_{h1} represents the set of points projected on the green region because of h_1 and S_{h2} represents the set of points projected on the green region because of h_2, then the candidate set as a result of $G(v)$ is $S_{h_1} \cap S_{h_2}$. Similarly, we are interested in the points that fall in the red region on both the projected lines. If R_{h_1} is the set of points that fall in the red region on

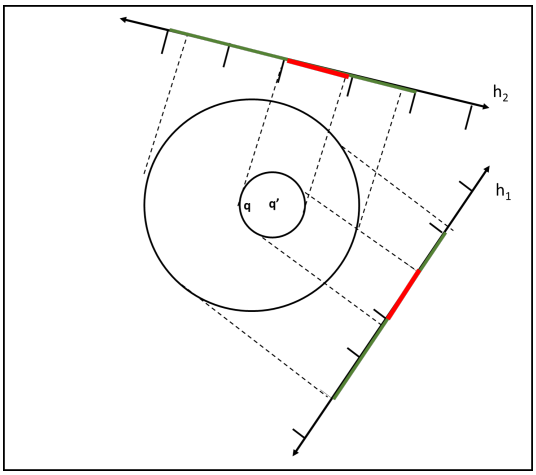


Figure 4.4: Single Hash Table Layer Formed by Intersection of k Hash Functions.

the projected line for $h1$ and R_{h_2} is the set of points that fall in the red region on the projected line for $h2$, then we would like to exclude the set of points belonging to $R_{h_1} \cap R_{h_2}$. Hence the final set of points that the resultant candidate set should contain is given by the following:

$$CandidateResults = \{S_{h_1} \cap S_{h_2}\} - \{R_{h_1} \cap R_{h_2}\} \tag{4.4}$$

In order to arrive at the two segments (marked by red and green) on the projected lines, we need to maintain two different bucket sizes. We also note that since the query q' is completely within q, the same is reflected in the projected segments, A larger bin w can contain a few smaller hash bins w'. The goal is to construct a hierarchical index structure such that the smaller hash bins are children of larger hash bins.

First, with the help of an example, the mapping of $G(v)$ to z-order space is discussed. Then, the same concept is extended to handle negative queries. Consider the Figure 4.5 with points v_1, v_2, v_3, and v_4 in two-dimensional space with x and y being the 2 co-ordinates. Let q be the query point. We see two lines, a_1 and a_2 on which the points are projected. Let these two lines represent the real lines on which the points are projected for hash functions h_1 and h_2. The two lines are then quantized into

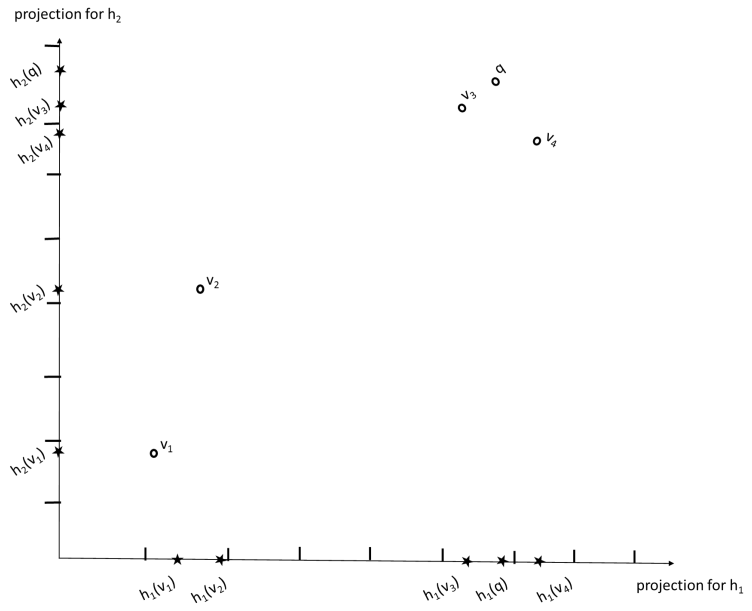


Figure 4.5: Projection Of Points v1, v2, v3, v4 and q on Two Hash Functions, Where k=2

equally sized bins. For instance, point v_1 falls into the bin number 1 of a_1 and bin number 1 of a_2 (where bin numbers start from 0). Similarly all the other points are projected on a_1 and a_2. This gives us with the following CHKs for the points namely $\{1,1\}$, $\{1,4\}$, $\{5,7\}$ and $\{6,6\}$ in the form of $\{h_1(v_i)h_2(v_i)\}$, where $i = 1, ..., n$ is the index of the points. We have now transformed the points in the original space to the new space.

Figure 4.6 represents the new space with the two co-ordinates being the $h_1(v)$ and $h_2(v)$ for any point v. This figure shows that $G(v_1)$, $G(v_2)$, $G(v_3)$ and $G(v_4)$ are points in the transformed space. As $G(v_1)$ falls in the cell formed by 001 in the $h_1(v)$ dimension and 001 in the $h_2(v)$ dimension, the z-code value of the point $G(v_1)$ is given by 000011(as discussed in Equation 3.1). Though similar technique of one-dimensional ordering has been used in [54] and [47], neither of them apply it to handle negative queries.

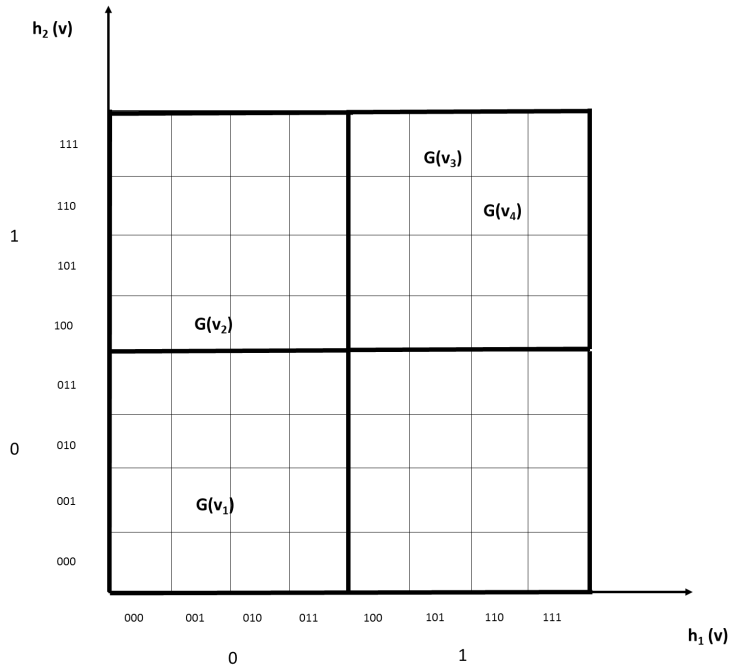


Figure 4.6: Points in the Reduced Space Where the Number of Dimensions is the Number of Hash Functions, k = 2.

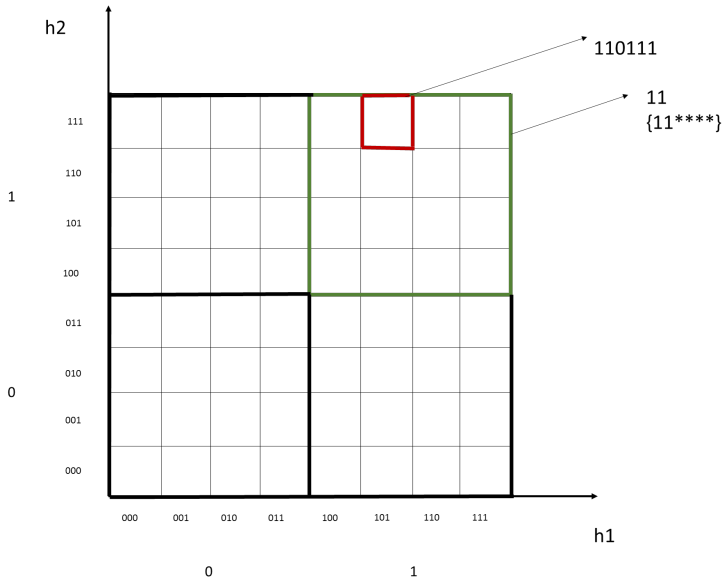


Figure 4.7: Hierarchy of Hash Functions and Z-order

With this information, we now see how mapping to z-order can be useful to handle negative queries. Figure 4.7 shows the relation between two different resolutions of z-order representation of CHKs. Let us assume that the width of the edge of each of the smaller cell is one unit. We can observe two different resolutions in the figure as follows:

- Let us consider the smaller resolution where both the dimensions are divided into two regions of width four units each, thus forming four quadrants. Since there are only two divisions in each dimension, we require one digit to label the cell in a dimension. Let us call this number the *precision* for the resolution. Let the number of hash bins formed for a hash function be given by U. Then U is the number of co-ordinates in a dimension, and its precision p of a resolution is given by,

$$p = log_2\ U \tag{4.5}$$

  Hence, for the smaller resolution, the co-ordinates in both the dimensions are 0 and 1. This is shown by bold lines in the figure.

- Next, we consider the larger resolution where the scale varies from 0 to 7 in both the dimensions where the width of the edge of each cell formed is 1 unit. The precision for this resolution is given by $log_2 8$ which is 3, thereby requiring 3 digits to represent each co-ordinate.

To uncover the relationship between the hierarchies of both the resolutions, consider the green square which represents a cell in the smaller resolution. This cell is labeled 11 by interleaving the bits of the two co-ordinates. The red square on the other hand, represents a cell in the higher resolution, labeled as 110111. All the cells in the higher resolution have the prefix of the cell it is contained within, in the smaller

resolution. 11, which is in the smaller resolution is the prefix of the cell 110111, which is in the higher resolution. We derive the following relationship between the labels of the cells (one-dimensional ordering) of smaller and higher resolutions.

Cell labeled 11 contains many more cells of higher resolutions. Using the label of the cell in smaller resolution, one can derive the labels of cells in higher resolution. Let p be the precision in the smaller resolution and p' be the precision in the higher resolution. Let x be the label of the cell in the smaller resolution. It contains $x'_0, ...x'_{n-1}$, where n is the number of higher resolution cells contained within a larger cell. The cell x'_0, with the smallest z-order labeling in the higher resolution contained within the cell x , in two dimension space is given by

$$x'_0 = x \ << \ (2 * (p' - p)) \tag{4.6}$$

Similarly, the cell x'_{n-1} with the largest z-order label in the higher resolution contained within the larger cell x is given by

$$x'_{n-1} = (x + 1) << (2 * (p' - p)) - 1 \tag{4.7}$$

With the help of Equation 4.6 and Equation 4.7, we can derive the relationship between the labels of the cells in different resolutions. The example above considers the case when the number of dimensions is two so that it is easy to visualize. The same can be applied for k number of dimensions. For k hash functions, the projected space is of dimension k. The above equations can be generalized to the following.

$$x'_0 = x \ << \ (k * (p' - p)) \tag{4.8}$$

Similarly, the cell x'_{n-1} with the largest z-order label in the higher resolution contained within the larger cell x is given by

$$x'_{n-1} = (x + 1) \ << \ (k * (p' - p)) - 1 \tag{4.9}$$

Algorithm 1 ComputeQueryRange

1: **procedure** ComputeQueryRange
2: $leftShift = k(p' - p)$
3: $zcode = ComputeHashCode(w)//This\ function\ first\ projects\ the\ points\ into\ lower\ dimension\ and\ computes\ zcode$
4: $QueryRange.begin = zcode << leftShift$
5: $QueryRange.end = (zcode + 1) << leftShift - 1$
6: return $QueryRange$

Algorithm 1 describes the method $ComputeQueryRange(...)$ that transforms the query range in the original space to the range in the one-dimensional ordering. The index structure is constructed with the highest resolution possible depending on optimum values for w'. A cell of smaller resolution is represented as a factor of a cell of higher resolution. This gives us an index structure where the bucket width grows in factors of 2^f, where f is an integer. Optimal w and w' values are chosen for the query point and the negative query point respectively. Hence, there can be a set of fixed bucket sizes for a particular data set depending on the data distribution. The one dimensional ordering along with the pointers to all the actual points, is stored in a tree data structure to perform efficient range searches.

4.2.3 At the Index Level

Multiple layers are used in LSH to avoid false negatives and improve the quality of the results. We construct L such hierarchical layers. Let R_{cand,l_i} be the result candidates at layer l_i, then the final set of candidates will be

$$CandidateResultSet = \bigcup_{l=1}^{L} R_{cand,l_i} \tag{4.10}$$

The hash functions in each of the $l * k$ hash functions are generated independently and randomly.

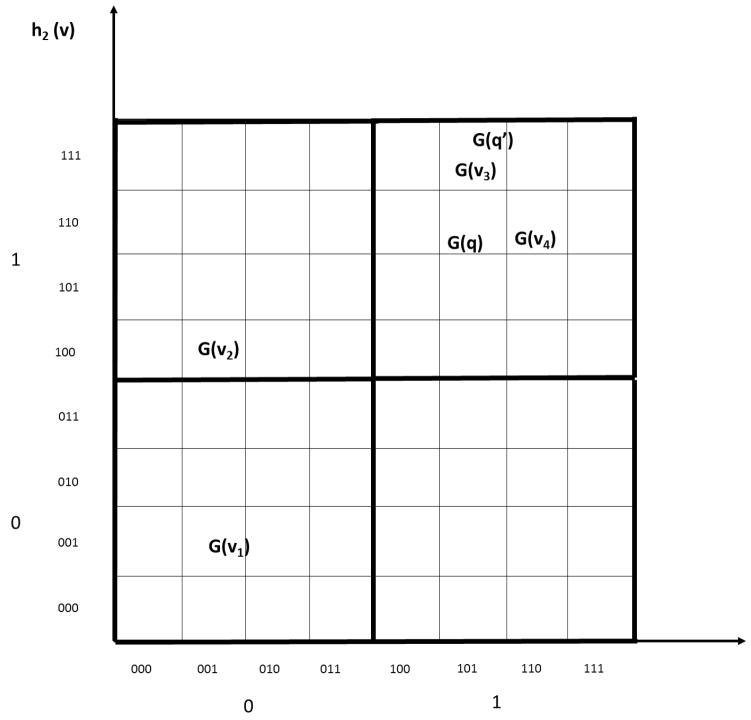


Figure 4.8: Points in the Reduced Space Where the Number of Dimensions is the Number of Hash Functions, k = 2 With Query Point.

4.3 Query Processing

This section discusses the efficient retrieval of results in the case of negative queries. Let us consider a larger query q with radius r and smaller query q' with radius r'. In the index building phase, the points were hashed into buckets of smaller resolution, i.e., buckets of width w. Depending on the query sizes, optimal w and w' are chosen for q and q'. An index structure of upto four levels is constructed to conduct experiments.

During the query processing stage, z-codes for both q and q' are computed. If the query range of q' is completely within q, then the $zcode(G(q))$ will be a prefix of $zcode(G(q'))$. The z-code of the query point is realized in terms of multiple cells of higher resolution. All the cells of higher resolutions are probed for candidates except

Algorithm 2 Get candidates from a single layer

struct $QueryRange\{begin, end\}$

Input:

- Main query , q
- Negative query , q'

procedure QUERYLAYER(LAYERNUMBER)

    $candidates = \varnothing$

    $queryRange = ComputeQueryRange(q)$

    $exclQueryRange = ComputeQueryRange(q')$

    for zcode:queryRange.begin to queryRange.end **do**

        if $zcode < exclQueryRange.begin$ *and* $zcode > exclQueryRange.end$ **then**

            $Candidates = Candidates \cap PointsHashedToZcode$

Algorithm 3 Get candidates from all the layers

procedure GETALLCANDIDATES

    $Candidates = \varnothing$

    for i:1 to number of layers, l **do**

        $IntCandidates = QueryRange(i)$

        $Candidates = Candidates \cup IntCandidates$

the cell with the label of z-code of q'. This would return all the candidate points which are satisfied by the query q without containing the points that satisfy the negative query q'.

We make use of Figure 4.8 to illustrate with an example. Using different w and w', we get two different z-codes. Cell labeled 11 for the larger query q and 110111 for the smaller query q'. Leveraging the fact that the z-code of the larger query is a prefix of smaller query, we can easily perform a range search on the one dimensional index structure for all nodes of the form 110000 to 111111 while excluding the results for 110111. This will not fetch any of the points that lie within the bucket containing points that fall within the excluded region. The pseudo-code for fetching candidates from a single layer and finally from all the layers together is outlined in Algorithm 2 and Algorithm 3 respectively. Algorithm 2 fetches candidates from a single layer

Algorithm 4 Post processing results

Input:

- A set of candidate results S, size of the result set n_s
- Main query radius, r
- Negative query radius, r'

procedure PROCESSRESULTS
    $Results = \varnothing$
    for i:1 to n_s **do**
        $distanceToQueryPoint = ComputeEuclideanDistance(q, S_i)$
        if $distanceToQueryPoint \leq r$ **then**
            $distanceToHoleQueryPoint = ComputeEuclideanDistance(q', S_i)$
            if $distanceToHoleQueryPoint > r$ **then**
                Add S_i to $Results$

in the presence of one exclusion region. If there are multiple exclusion regions, range search is performed excluding candidates from all the exclusion regions.

4.3.1 Post Processing

Though we have eliminated most of the points that fall in the excluded regions, since LSH is an approximation scheme, there is a need to filter out the results from the generated candidate set. To accomplish this step, Euclidean distance is computed between the query point and every other point to check whether the point is within the bounds of the main query. Then, Euclidean distance is computed between the negative query point and the points within the main query to further rule out any false positive due to the excluded region. The pseudo-code for the same is outlined in Algorithm 4.

4.4 Theoretical Analysis

In Section 3.2.2, we discussed the quality guarantee for the traditional LSH. The proposed algorithm modifies the existing LSH to handle excluded queries. Hence, the

theoretical probability guarantees that hold for the traditional LSH, do not remain the same. Since the excluded regions are also eliminated using approximation, the probabilities for a point to collide with the query point also changes.

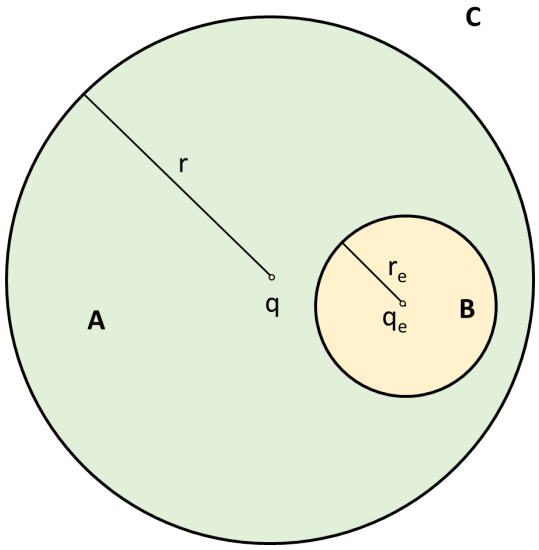


Figure 4.9: Query Region and Excluded Region. Region A is the Main Region of Interest Shown in Green. Region B is the Region to Exclude from the Main Region of Interest. Region C is Any Region Outside of Range r^+ from Query Point, q.

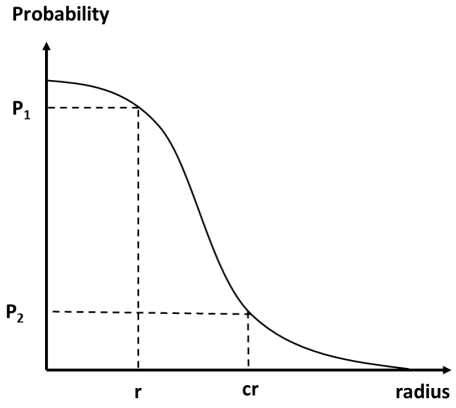


Figure 4.10: Impact of Radius on Probability for a Fixed Bucket Size, w.

As illustrated in Figure 4.9, let the main region of interest be defined by a query

point, q and a radius of r. Let q_e be the query point for the excluded region and r_e, the radius of the excluded region. Let us denote the region in green as A, which is the main area of interest. Let the region in orange be denoted by B which is the region to be excluded. Let C be the region outside of A, which we are not interested in. Since we make use of another hash function for excluded region, we make use of probabilities of both the queries to derive a combined success probability.

Using Definition 3.1 of LSH, let us define two hash families to answer the main query and excluded region respectively. Let (r, cr, P_1^+, P_2^+) be the family of hash functions to answer the main query. Similarly, let $(r^-, c^-r^-, P_1^-, P_2^-)$ be the family of hash functions to answer the excluded region query. Figure 4.11 represents the two hash families. Let $h(v)$ be the the hash function for the main query and let $h'(v)$ be the hash function for the excluded region. Recall that the individual hash functions in the main query and exclusion query are different because of the bucket sizes. Also, an object in the main query is returned if all the hash functions in a layer collide with q. Similarly, an object in the excluded region is eliminated, if the object collides with q_e in all the hash functions in a layer. Using these families of hash functions, we can derive the probability with which an object within the region of interest is returned.

We see from Figure 4.10 that, for a fixed w, as the radius increases, the probability of the point within the radius being returned, decreases. In the case of main query, for a given pair of r and cr, the radius is chosen to be close to r in order to avoid misses. This also gives rise to false positives (objects that lie in region C), but those can be removed during post processing. However, in the case of the excluded region, with an exclusion radius of r_e, if the hash function for excluded region is chosen such that r^- is equal to r_e, the probability of elimination of points within B increases. However, this has a side effect of eliminating the good candidates (false positives in the case of exclusion query) outside of B. This results in misses from the actual

region of interest, A, which needs to be avoided. Hence, we define probabilities with which an object is returned, depending on the region it occurs in.

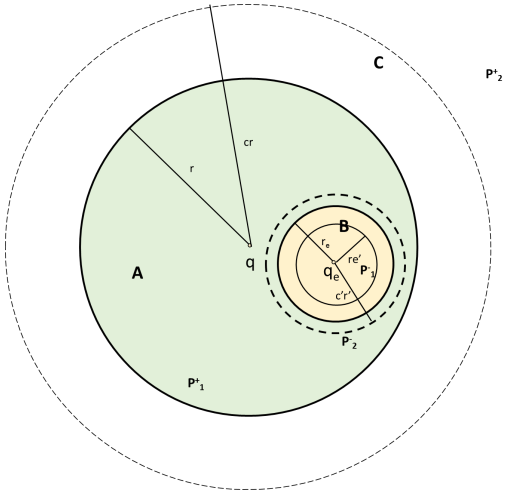


Figure 4.11: Regions A, B and C Along with Probability Guarantees for Queries with Different Radiuses.

The success of the proposed approach can be assessed using the following probabilities:

1. Probability, ψ_l^+, of returning an object in region A
2. Probability, ψ_l^- of not returning an object in region B or C

Probability of returning an object in region A

In a given layer, let $G(v)$ be the compound hash function for the main query and let $G'(v)$ be the compound hash function for the exclusion query. Probability of returning the points within the region A, relates to the recall of the results. This depends on the probabilities on both $G(v)$ and $G'(v)$ as follows:

- Prob[An object in region A is returned by $G(v)$]. The probability that an object in A is returned by $h(v)$ is P_1^+. Probability that a layer of hash functions returns

the object in A is P_1^{+k}.

- Prob[$G(v)'$ does not eliminate a point in region A]. We need to choose the r^- and c^-r^- such that the probability of collision of a point in A with q_e is minimized. If we choose c^-r^- such that, it is aligned with r_e, the probability that a point in A collides with q_e is P_2^-. The probability of not eliminating a point in A translates to probability of not returning a point in A by the excluded region hash function. The probability of not returning a point within A by at least one of the hash functions (of excluded regions) in a layer is $(1 - (P_2^-)^k)$.

For a given layer, if we assume that the hash functions are independent of to each other, the overall probability of returning a point that is in the region A is $P_1^{+k}.(1 - (P_2^-)^k)$.

If ψ^+ is the probability of returning correct results in the entire index, it can be obtained by computing the probability of returning correct results in at least one of the l layers and is given by,

$$\psi^+ = 1 - (1 - P_1^{+k}.(1 - (P_2^-)^k))^l \tag{4.11}$$

Probability of eliminating an object in region B or C

Probability that the negative candidates are not returned relates to the precision of the returned results. This probability depends on the following:

- Prob[$G(v)'$ eliminates all the points in B]. This measures the probability of exclusion hash function to correctly eliminate the irrelevant results. In an ideal case, the value of c^- will be very close to 1 ($c^- = 1 + \epsilon$, where ϵ is close to 0). This implies the probability of collision of point inside of r^- with query point is almost equal to the probability of collision of a point inside of r^-c^- with the

query point. Probability that a point within B collides with q_e is P_1^-. Hence, probability of $G(v)'$ to eliminate irrelevant candidates for the entire CHK is given by $(P_1^-)^k$.

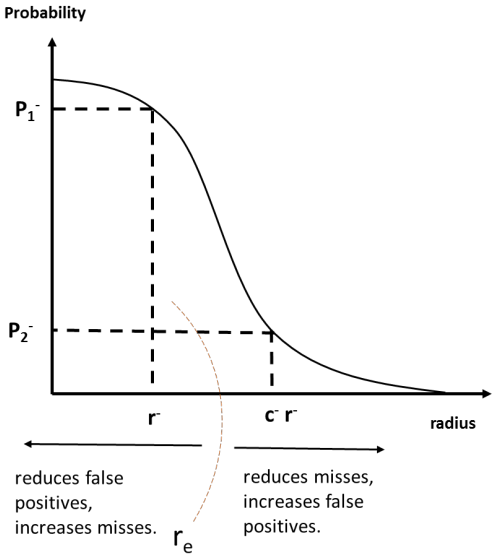
- Prob[$G(v)$ does not return a candidate in C].The probability that the query collides with a point in region C, in one hash function is P_2^+. Probability that the query collides with a point in region C in all hash functions in a layer is $(P_2^+)^k$. Probability that the query point does not collide with a point in region C in at least one hash function is $1 - (P_2^+)^k)$.

For a given layer, probability that incorrect results are not returned is given by $P_1^{-k}.(1 - (P_2^+)^k))$

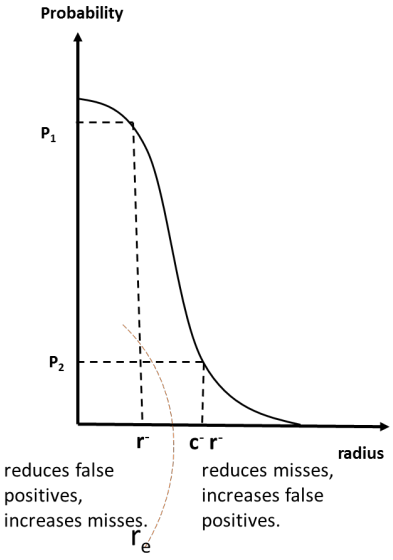
If ψ^- is the probability of not returning incorrect results in the entire index, it can be achieved by computing the probability of not returning incorrect results in all the layers.

$$\psi^- = ((P_1^-)^k.(1 - (P_2^+)^k))^l \tag{4.12}$$

The success of the algorithm depends on maximizing the probabilities in the Equations 4.11 and 4.12. For a given value of r_e if we choose the index parameters such that r_e is close to r, then we eliminate most of the candidates within the excluded region. On the other hand, we introduce a lost of misses because of eliminating candidates from region A. Figure 4.12(a) shows that if r_e is chosen close to r^-, then misses are increased, whereas if r_e is chosen close to c^-r^-, then false positives increase. For the given radii of the main query and exclusion query, and the values of w and w', we choose the value of k that maximizes both these probabilities ψ^+ and ψ^-. Increasing ψ^+ avoids misses in the results, thereby increasing recall of the returned results. Increasing ψ^-, decreases false positives, thereby reducing post processing cost. The



(a) Effect Of Choosing Hash Function For Excluded Region



(b) Designing Hash Functions Such That c^- Is Close To 1

Figure 4.12: Effect Of Choosing Hash Function For Excluded Region

ideal selection of hash functions would be such that c^- is very close to 1. This implies that r^- and r^-c^- are very close. All the objects within r^- will be eliminated with high probability and all objects outside of r^-c^- are not eliminated in the final result.

Chapter 5

EXPERIMENTS AND RESULTS

This section discusses the experimental setup, the system setup, the data sets used in evaluation, the evaluation criteria used, and the analysis of results.

5.1 Experimental Setup

Experiments are conducted on three data sets namely Color, Aerial and Uniform data.

5.1.1 Color Dataset - Real Data

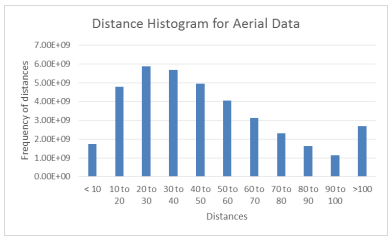
Corel dataset is a collection of 68,040 real images [2]. The data used are the feature vectors extracted from these images. More specifically, the color dataset consists of color histograms of these images that represents density of colors in the image. Each object in this data set is of 32 dimensions. The values in each of the co-ordinates of the data is between 0 and 1.

5.1.2 Aerial Data - Real Data

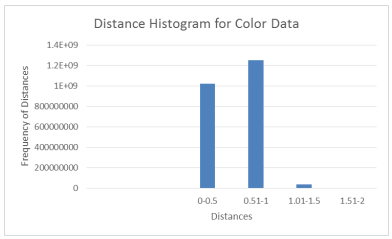
Aerial data consists of large aerial photographs. It contains 275,465 data objects which are texture descriptors. Each feature vector is of 60 dimensions [1]. The values in each co-ordinate of the data varies from 0.008 to 270.

5.1.3 Uniform Data

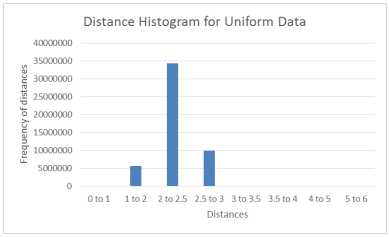
To analyze the behavior of the algorithm in uniform distribution, synthetic data is generated. The size of the data set is 100,000 data objects. The value of each co-



(a) Aerial Data



(b) Color Data



(c) Uniform Data

Figure 5.1: Distance Histogram of Datasets

ordinate of a data object is a number that is generated uniformly at random. Each data object contains 32 dimensions. The values in each of the co-ordinates range between 0 and 1.

Figures 5.1(a), 5.1(b) and 5.1(c) show the distance histograms of the datasets used in the experiments.

5.2 System Setup

Experiments were executed on a Microsoft Windows(R) 7 (x64) machine having, Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz processor. The algorithm was implemented in C++ using Microsoft Visual Studio environment.

5.3 Comparison Algorithm

Since none of the existing LSH techniques handle queries with exclusions, a naive method is implemented using LSH. In the naive approach, an LSH index structure is implemented as efficiently as possible. Points that are within a range of r from query q, are computed using the naive LSH index. These candidate results would also contain points that are within the exclusion region. In the post processing step, Euclidean distance is computed from the main query point to the object, to check if it is within a distance of r. If the point lies within the query range of r, then Euclidean distance is computed with the exclusion query point q'. If the point lies within the exclusion radius of r', then such a point is eliminated. In short, the candidates for the main query are fetched. In the post processing step, the points in the excluded region are eliminated. For simplicity, let us call this approach LSH-Naive and proposed approach, LSH-E (LSH with Excluded Regions).

5.4 Evaluation Criteria

Next, we discuss the various evaluation techniques that were employed to evaluate the accuracy and performance of the algorithm.

5.4.1 Time Taken to Process Queries

The efficiency of the algorithm is evaluated based on the time taken to retrieve the results. Time taken represents the combined time taken to fetch the candidate results, and to post process them. Percentage gain with respect to time taken in LSH-Naive and LSH-E is computed. Let time taken by LSH-Naive be $t_{LSH-Naive}$ and time taken by LSH-E be t_{LSH-E}. Percentage gain in time taken is computed by

$$gain = \frac{t_{LSH-Naive} - t_{LSH-E}}{t_{LSH-Naive(for\ 0\ excludedregions)}} \tag{5.1}$$

Percentage of time gain is computed for zero excluded regions, one excluded region and two excluded regions. All the gains are normalized with respect to time taken by zero excluded regions. The gain in time is computed by varying the parameters - number of hash functions k, number of layers, l and radius of the excluded region, r'.

5.4.2 Candidates Savings Vs. Recall Loss

The proposed algorithm reduces the number of candidates fetched, thereby reducing post processing time. In order to ensure that the results returned are relevant, we measure both the Candidates Savings, as well as Recall Loss. Candidates Savings is the drop in the number of candidates that need to be post-processed. Since the proposed algorithm introduces an approximation to eliminate irrelevant candidates, there is a slight loss in terms of recall. Recall is computed as a ratio of the number of correct results returned by the LSH algorithm to the actual number of correct results. In the experiments, both the gain in terms of candidates and loss in terms of recall are discussed. Let $Recall_{LSH-Naive}$ be the recall in the case of LSH-Naive and $Recall_{LSH-E}$ be the recall in the case of LSH-E, then loss in recall is as follows:

$$Loss\ in\ Recall = \frac{Recall_{LSH-E} - Recall_{LSH-Naive}}{Recall_{LSH-Naive}} \tag{5.2}$$

Let $Cand_{LSH-Naive}$ be the number of candidates returned by Naive LSH, and $Cand_{LSH-E}$ be the number of candidates returned by LSH-E, then gain in terms of candidates drop is given by the following:

$$Candidates\ Savings = \frac{Cand_{LSH-Naive} - Cand_{LSH-E}}{Cand_{LSH-Naive}} \tag{5.3}$$

The goal of this research is to achieve a gain in Candidates Savings while maintaining a low Loss in terms of Recall. The experiments are executed by varying the parameters k, l and r.

Parameter	Values
Datasets	Uniform, Color, Aerial
k	7, **8**, 9, 10, 11
l	3, 4, **5**, 6, 7
No. of excluded regions	0, 1, 2
ratio of volume of excluded region to query region	**1/4**, 1/10, 1/20, 1/50

Table 5.1: Parameter and Values Used in Experiments (defaults in bold)

5.5 Results

The three data sets have different distributions and this has an influence on the choice of various parameters. The distance histogram of the distances between points for the Uniform dataset has most of the points between 2 units and 2.5 units. Hence, r is chosen to be 2.3. We select a radius for the excluded region such that the size of the excluded region is one-fourth the size of the main query. Therefore, the value of r'^{-} for Uniform dataset is 2.2 (32 dimensions). For the Uniform dataset, the parameter w for the main query is set at 3 and we use the next smaller value of w' in the hierarchy, that is 1.5, for the excluded regions. These values of bucket size provide recall guarantees of over 0.7. For the Color dataset, w for the main query is set as 0.8 and w' for the excluded region is set as 0.4. The chosen radius of main query is 0.6 and radius of excluded region r'^{-}, is 0.574. For the Aerial dataset, w is chosen to be 80 and w' is chosen to be 40. The chosen radius values of r and r'^{-} are 50 and 48.75 respectively. As discussed in the theoretical analysis Section 4.4, the w' for excluded region is chosen such that, the probability of not returning a good candidate because of excluded region, is minimized. The radius of the exclusion region is aligned to the $c^{-}r^{-}$ radius which determines P_2.

Table 5.1 gives the list of parameters that were varied to carry out the experiments. The default values for k, l, and volume of excluded region to query region ratio are 8, 5 and 1/4 respectively. The w' values chosen for the excluded region hash functions are selected such that, the radius of excluded region is aligned with the miss probability of $c^{-}r^{-}$. Choosing a w' in such manner avoid misses in the final result.

5.5.1 Time Taken

Impact of k on Time Taken

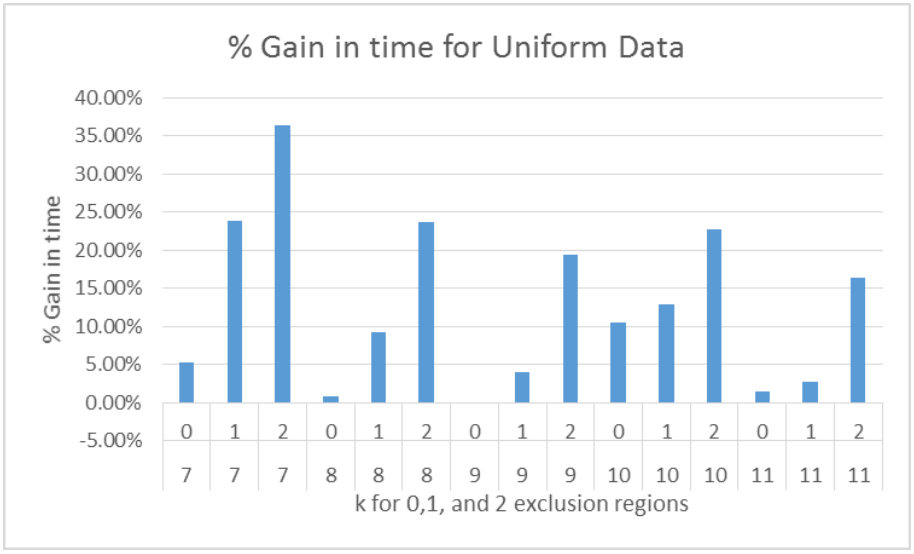


Figure 5.2: Percentage Gain in Time Taken for Values of k in the Presence of 0,1 and 2 Excluded Regions Observed in Uniform Data. $l = 5$

Figures 5.2, 5.3, and 5.4 show that the proposed algorithm has consistent gain compared to naive LSH, when there are one or two excluded regions. In the case of zero excluded regions, the gain is almost equal to zero. The minimal gain and loss in the case of zero excluded regions is due to average computation times across multiple runs. The experiment results show that, as the value of k increases, the time gain for LSH-E with respect to LSH-Naive reduces. As the number of hash functions

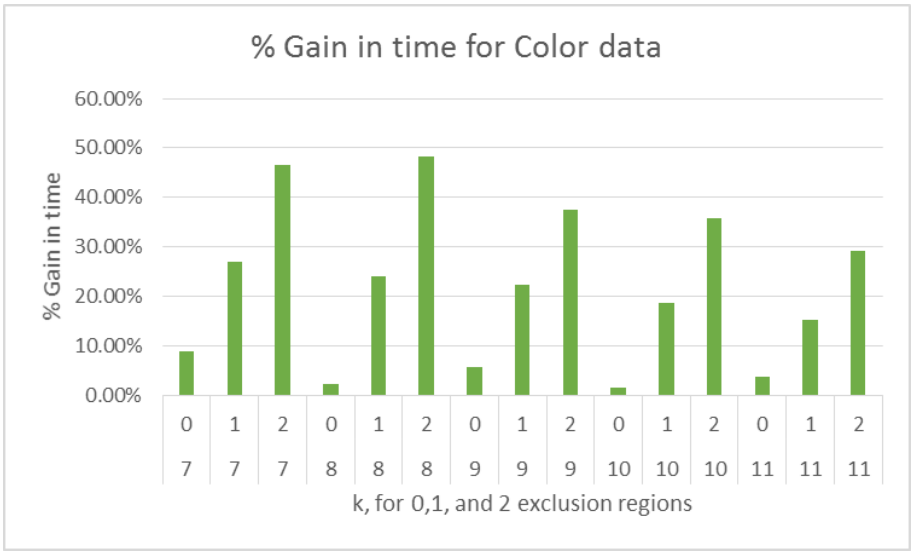


Figure 5.3: Percentage Gain in Time Taken for Values of k in the Presence of 0,1 and 2 Excluded Regions Observed in Color Data. $l = 5$

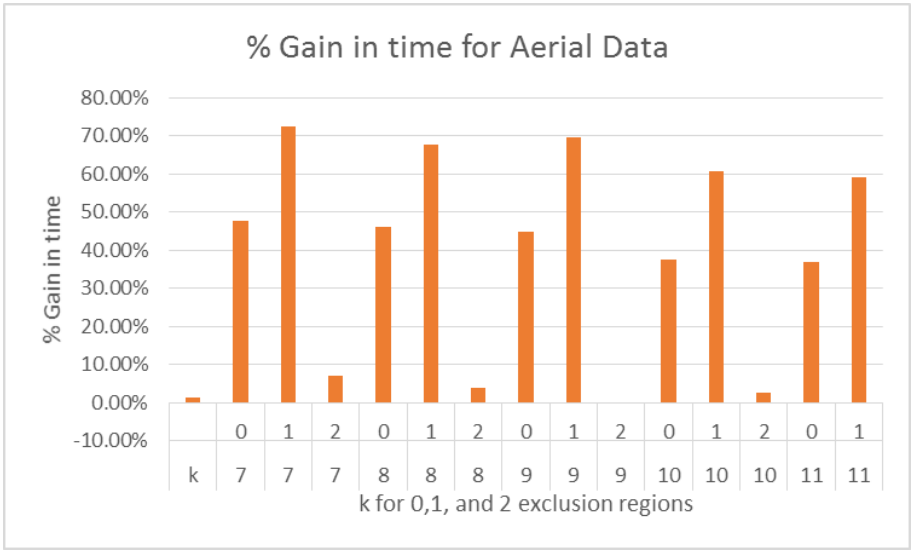


Figure 5.4: Percentage Gain in Time Taken for Values of k in the Presence of 0,1 and 2 Excluded Regions Observed in Aerial Data. $l = 5$

k increases, the hash function for answering the main query becomes stricter. This reduces the number of candidates in the first place. On the other hand, the number of candidates to be eliminated because of the excluded region also reduces, since the

hash function for the excluded region query becomes stricter. This leads to reduced benefits of elimination of excluded region in the case of LSH-E.

The time taken to post process the second exclusion region is higher than that for the queries with one exclusion region. This is because the Euclidean distance computation is done twice for two exclusion regions. With reduced number of candidates to post process in the case of LSH-E, the time gain in case of two exclusion regions is more than queries with one excluded region.

Impact of l on Time Taken

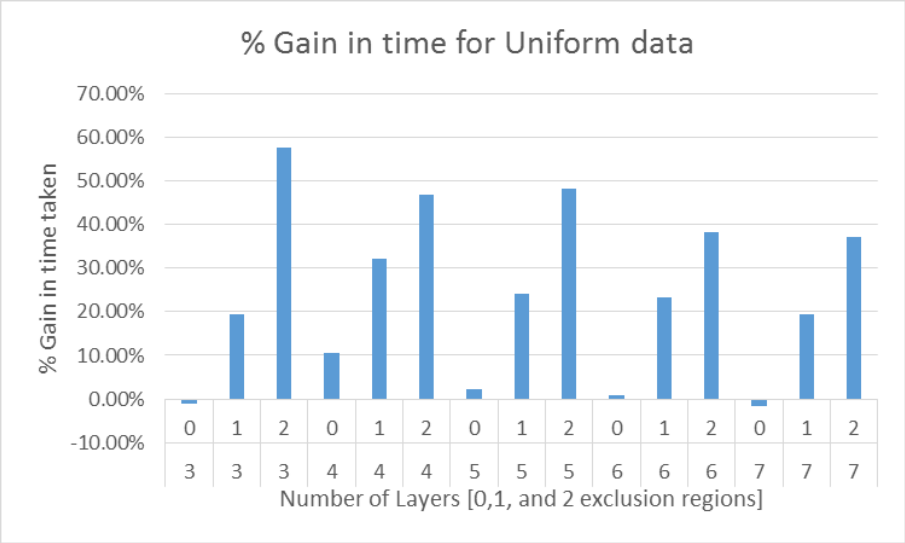


Figure 5.5: Percentage Gain in Time Taken for Values of l in the Presence of 0,1 and 2 Excluded Regions Observed in Uniform Data. $k = 8$

Figures 5.5, 5.6, and 5.7 show that, LSH-E shows an improvement with respect to time taken by LSH-Naive. In the case of zero excluded regions, both the methods take almost the same time.

As the number of layers increase, the rate of increase in terms of candidates is low. Each layer adds new candidates which might already have been returned by the

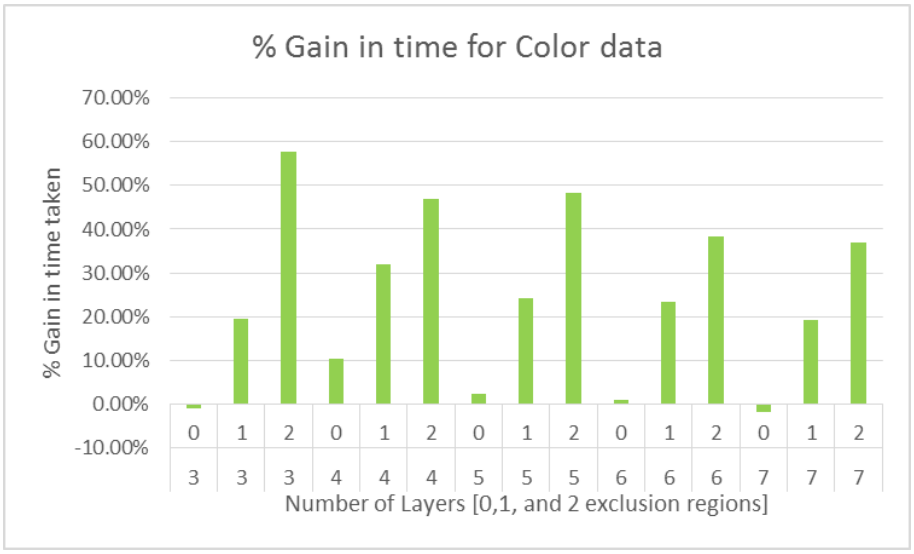


Figure 5.6: Percentage Gain in Time Taken for Values of l in the Presence of 0,1 and 2 Excluded Regions Observed in Color Data. $k = 8$

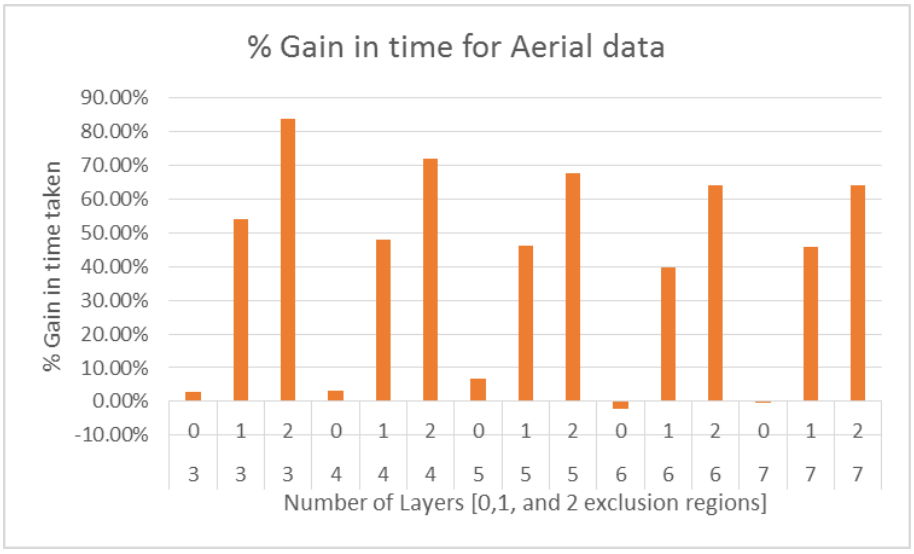
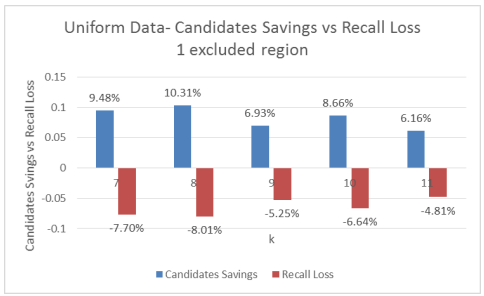
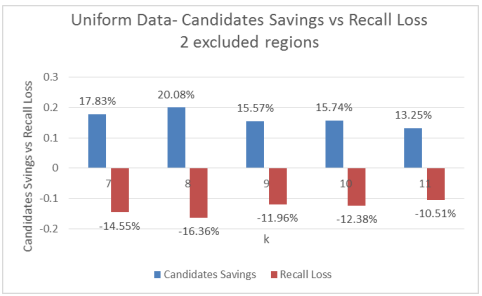


Figure 5.7: Percentage Gain in Time Taken for Values of l in the Presence of 0,1 and 2 Excluded Regions Observed in Aerial Data. $k = 8$

other layers. Similarly, the rate of increase of number of candidates for elimination with addition of new layers is low. This reflects in lesser gain in terms of processing time.

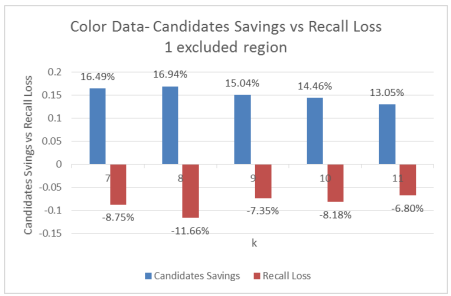


(a) One Excluded Region

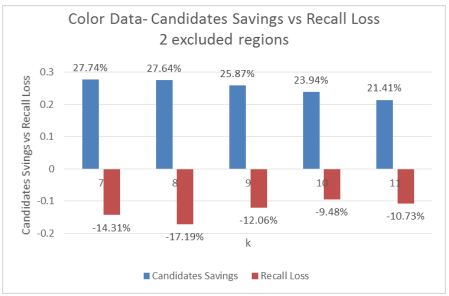


(b) Two Excluded Regions

Figure 5.8: Effect of k on Candidates Savings and Recall Loss on Uniform Data



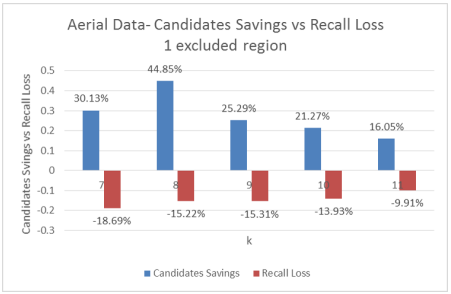
(a) One Excluded Region



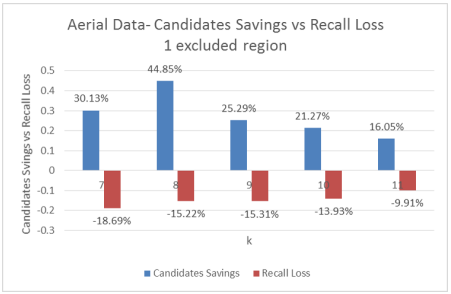
(b) Two Excluded Regions

Figure 5.9: Effect of k on Candidates Savings and Recall Loss on Color Data

In the case of time gain, we notice that the gain in terms of two excluded regions is more than the gain in the case of one excluded region. Firstly, more candidates are pruned in the case of two excluded regions. Of the remaining candidates to be post-processed, Euclidean distance needs to be computed with the main query and the excluded region query points. This leads to a high gain in terms of time taken for LSH-E with respect to LSH-Naive.



(a) One Excluded Region



(b) Two Excluded Regions

Figure 5.10: Effect of k on Candidates Savings and Recall Loss on Aerial Data

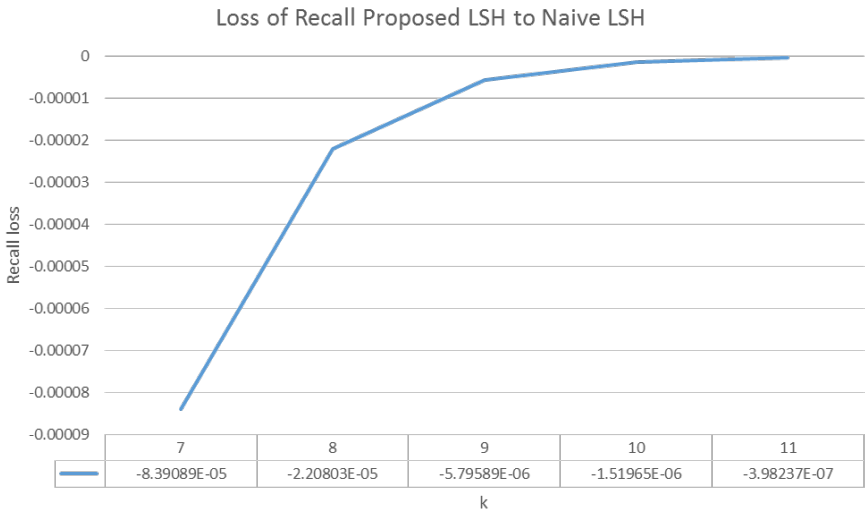


Figure 5.11: Impact of Recall on Varying k

5.5.2 Candidates Savings vs Recall Loss

Impact of k on Candidates Savings and Recall Loss

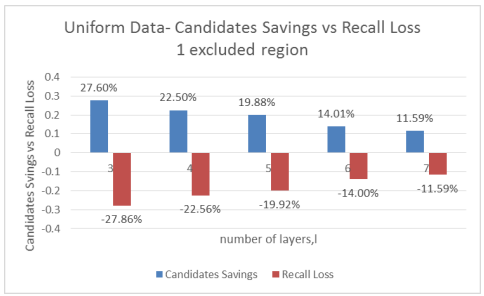
Figures 5.8, 5.9, and 5.10 show the impact of changing k on the candidates savings and the recall loss. It can be observed that the Uniform data shows the least gain compared to the real datasets. In the case of Uniform data, the points are uniformly distributed and hence the variance of distance between any two points is very low.

Whereas in the case of real data, the points seem to be clustered. In the case of Uniform data in high-dimension, all the points are at almost equal distances to each other. In the process of elimination of points in the excluded region, the points within the main query of interest also get eliminated. Due to this, it can be observed that, the gain in terms of candidates saved is almost equal to the loss in recall in the case of Uniform data. In case of real data, the impact of elimination of exclusion region is less on the points in the main region of interest. Hence, the excluded regions are removed more effectively in the case of LSH-E.

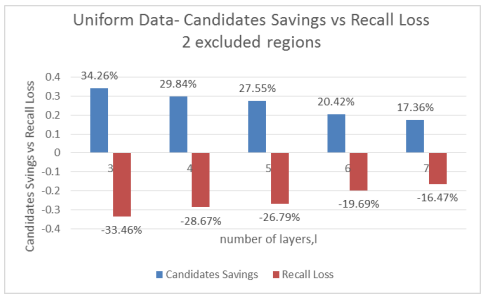
In the case of zero excluded regions, the gain in terms of candidates is zero. To measure the gains in the case of excluded regions, the hash functions for LSH-Naive and LSH-E are chosen to be the same for the base case of zero excluded regions. As the number of hash functions increases, the hash function for main query becomes more stricter, which reduces the number of candidates. As discussed in Section 5.5.1, the excluded region query also becomes more strict but excludes fewer candidates than the decrease in candidates in the main query. This explains the decrease in candidates savings. As k increases, the main query returns lesser candidates both in the case of LSH-naive and LSH-E. In the case of LSH-E, lesser number of excluded region candidates are eliminated. Therefore, the loss in recall reduces.

Impact of l on Candidates Savings and Recall Loss

Figures 5.12, 5.13, and 5.14 show the impact of changing l on the candidates savings and the recall loss. As the number of layers increase, the number of candidates increase due to the hash functions of the main query. As candidates increase, recall in the LSH-Naive increases. Whereas, in the case of excluded region queries, as the number of layers increase, the points of interest become candidates for elimination in the case of LSH-E. For a candidate to be eliminated, it needs to be eliminated by

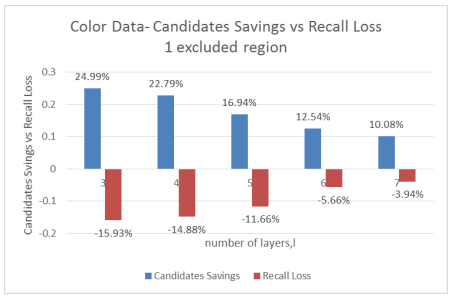


(a) One Excluded Region

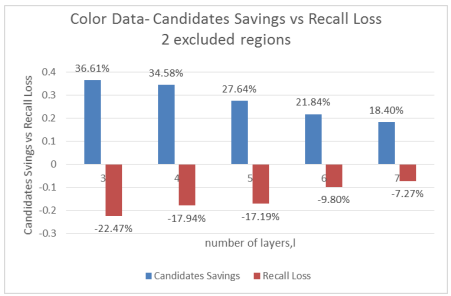


(b) Two Excluded Regions

Figure 5.12: Effect of l on Candidates Savings and Recall Loss on Uniform Data

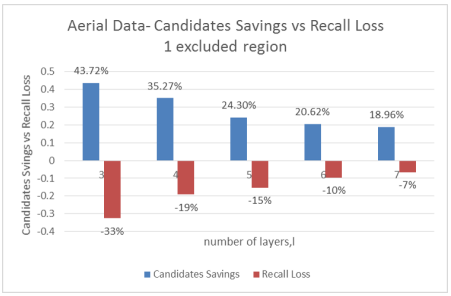


(a) One Excluded Region

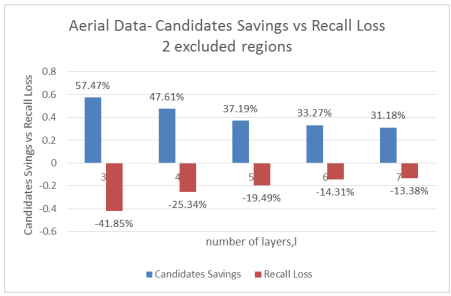


(b) Two Excluded Regions

Figure 5.13: Effect of l on Candidates Savings and Recall Loss on Color Data



(a) One Excluded Region



(b) Two Excluded Regions

Figure 5.14: Effect of l on Candidates Savings and Recall Loss on Aerial Data

all the layers. A candidate eliminated in one layer can re-appear as a candidates, if another layer did not eliminate it. Hence, the number of candidates returned increases

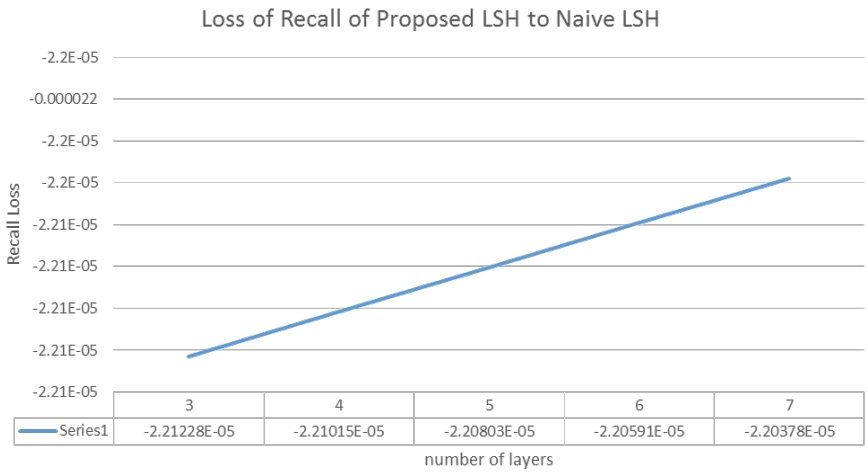
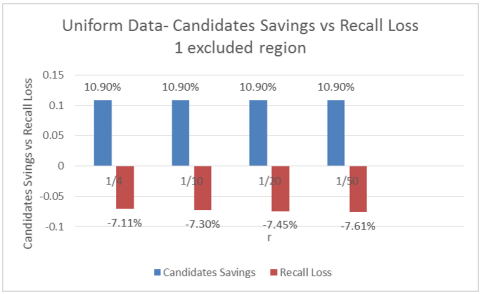


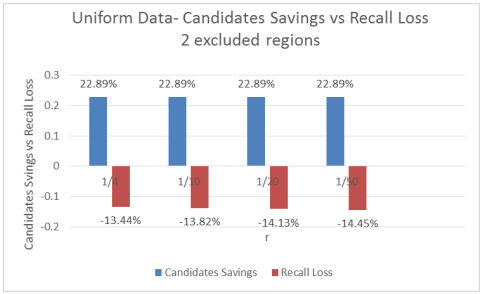
Figure 5.15: Impact of Recall on Varying Number of Layers

and the ratio of candidates saved decreases. With the increased number of candidates, the recall loss is minimized. It can be observed that the real data show more gains with respect to candidates savings versus recall loss than in the case of Uniform data, due to the distribution of points in the datasets. In the case of zero excluded regions, the number of candidates in the case of LSH-Naive and LSH-E are the same. Hence there is no candidates savings and no recall loss.

Figures 5.11 and 5.15 show the trends of ratio of Recall Loss in LSH-Naive to LSH-E for varying k and l respectively. These results are collected using the theoretical analysis using Uniform dataset. The decreasing recall loss shows a similar trend as our experimental results. However, theoretical ratios show a lower value of loss compared to experiments. This could be because of the distribution of data points. For a given exclusion radius, more points get eliminated because of the exclusion query radius. The probability of exclusion outside of $c^{-}r^{-}$ (defined in Section 4.4) is more in reality when compared to the theoretical analysis.

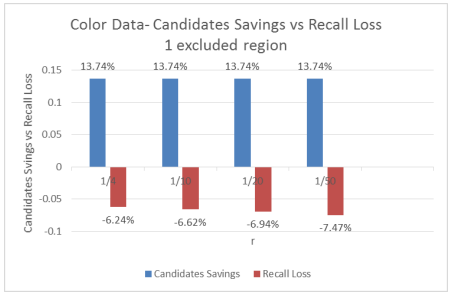


(a) One Excluded Region

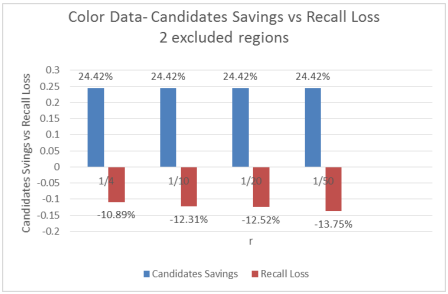


(b) Two Excluded Regions

Figure 5.16: Effect of r on Candidates Savings and Recall Loss on Uniform Data



(a) One Excluded Region



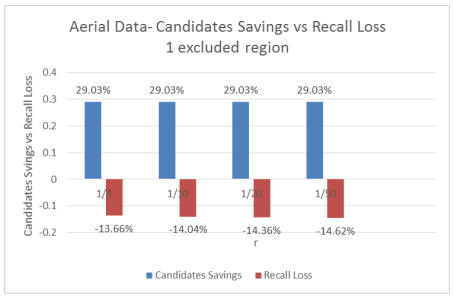
(b) Two Excluded Regions

Figure 5.17: Effect of r on Candidates Savings and Recall Loss on Color Data

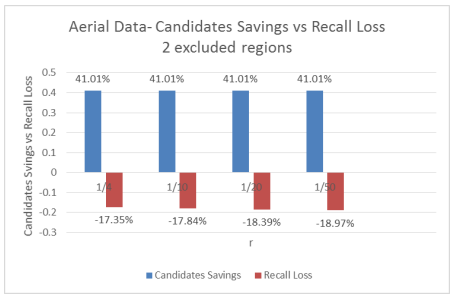
Impact of r_e on Candidates Savings and Recall Loss

The experiments were executed keeping the default parameters of k and l. The radius of the exclusion region was changed to reflect the volume ratios of 1/4, 1/10, 1/20, and 1/50.

Figures 5.16, 5.17, and 5.18 show the impact of changing the excluded region radius. In these experiments, the default parameters are set to the values corresponding to parameters of 1/4 volume. The values of w and w' are set for 1/4 volume values and are not changed while varying the exclusion radius. The number of candidates remain the same because the parameters used in the index are the same. Same num-

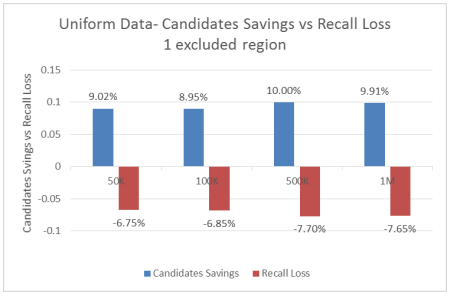


(a) One Excluded Region

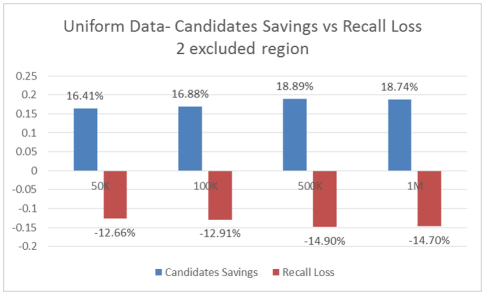


(b) Two Excluded Regions

Figure 5.18: Effect of r on Candidates Savings and Recall Loss on Aerial Data



(a) One Excluded Region



(b) Two Excluded Regions

Figure 5.19: Effect of Density of Data Distribution in Uniform Data

ber of candidates are fetched for both LSH-Naive and LSH-E. In the case of LSH-E, with the excluded region hash function, a set of candidates belonging to the main query are also eliminated. If the excluded region radius is small, the misses caused due to elimination of candidates from the main query is significant. When the radius of excluded region increases, some of these elements from the main query which were eliminated are not required to be returned any more. This improves the recall in the case of LSH-E when the radius of exclusion region is larger.

5.5.3 Impact of Density of Data Distribution in Uniform Data

To analyze the impact of density of data in the case of Uniform data, datasets were generated with sizes 50,000, 100,000, 500,000, and 1,000,000, each having 32 dimensions. Each co-ordinate of the object was randomly generated with values between 0 and 1. The parameters, bucket size, number of hash functions and number of layers were set to be default (as given in Table 5.1). Figure 5.19(a) and 5.19(b) show the Candidates Savings and the Recall Loss for the different densities in the case of one excluded region and two excluded regions. As the density increases in case of Uniform data, the number of pairs of points having smaller distances increases. Also, the number of pairs of points having distances in the range of 2.0 to 2.5 units also increases proportionately. With increased density, the number of objects that get eliminated due to the excluded regions increases. But good candidates also get eliminated proportionately, which decreases recall. Therefore, we can observe a slight increase in the candidates savings, at the same time a slight increase in recall loss.

5.5.4 Queries with Large Number of Small-Sized Excluded Regions

This section analyzes the behavior of a single large excluded region in comparison with a large number of small-sized excluded regions. For every dataset, 100 query sets are created such that, each query set has a single main query and 100 small excluded region queries which are 1/400 in volume compared to the main query. In the case of excluded regions with smaller hole size, w' for the excluded region query is chosen to be $w/4$. Whereas in the case of excluded region of volume 1/4, w' is chosen to be $w/2$. A smaller w' is chosen so that the elimination of good candidates is minimized for small-sized excluded regions. Figure 5.20 shows the behavior in the case of Color dataset. Due to the clustering of data points, a query with 100 excluded regions

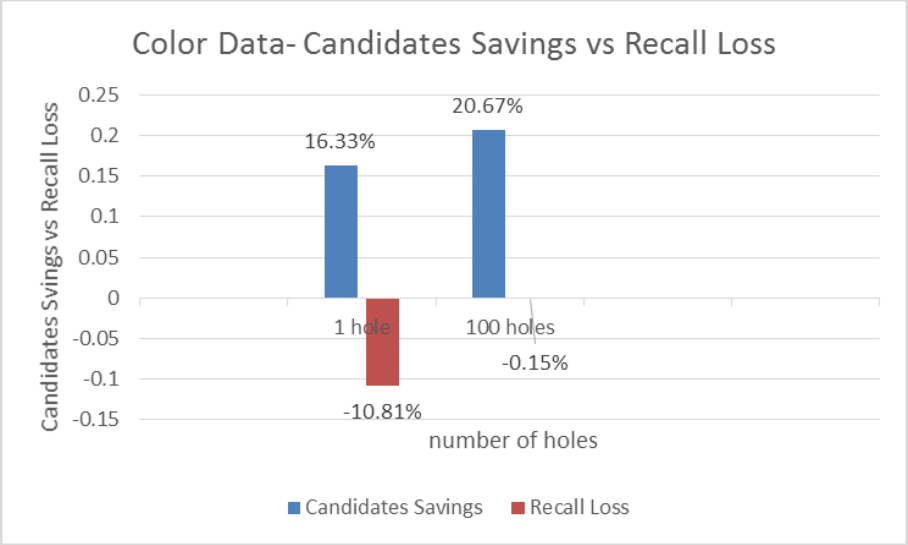


Figure 5.20: Queries with One Large Excluded Region of 1/4 Volume Vs. Queries with 100 Excluded Regions of 1/400 volume

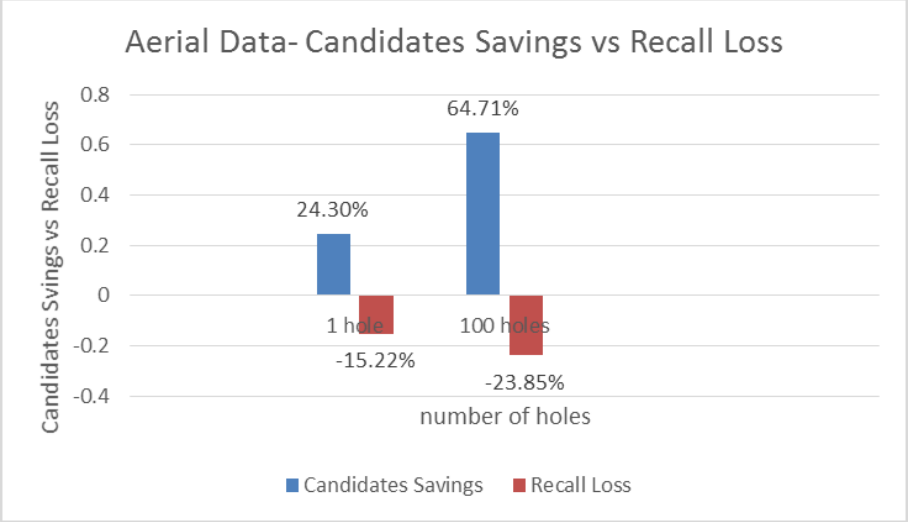


Figure 5.21: Queries with One Large Excluded Region of 1/4 Volume Vs. Queries with 100 Excluded Regions of 1/400 Volume

returns very few points as a final result. In the case of LSH-Naive, the number of candidates remain the same and the excluded region points are eliminated in post processing step. Whereas, in the case of LSH-E, more number of points are eliminated

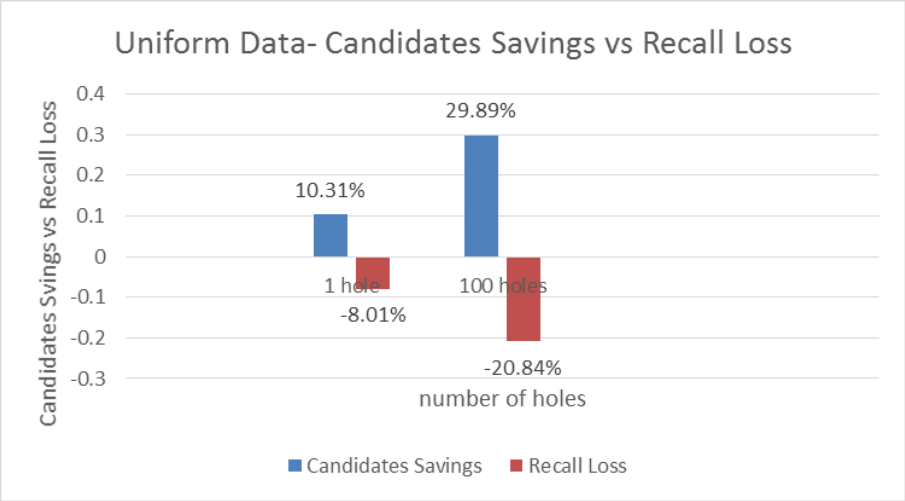


Figure 5.22: Queries with One Large Excluded of 1/4 Volume Vs. Queries with 100 Excluded Regions of 1/400 Volume

due to 100 queries and the actual results are not eliminated. Hence, the candidates savings is high and loss in recall is extremely low. Of the 100 query sets, 37 of them returned 0 as the final result. In those queries, LSH-E fetched 9% candidates less compared to LSH-Naive. In the case of Aerial dataset too, we see a high gain in terms of candidates saved. However, some of the good candidates are also lost due to elimination. But the ratio of candidates savings to loss in recall is high as seen in Figure 5.21. In the case of Uniform data, the ratio of gain in terms of candidates and loss in recall is the least. With uninteresting candidates being eliminated, good candidates are also lost in the case of Uniform data as shown in Figure 5.22. Due to the distribution of Uniform data in high-dimension, a lot of points lie in between r^- and r^-c^- which may or may not get eliminated. In the case of real data, the proposed approach proves to be more beneficial for candidates savings in case of multiple small-sized excluded regions.

Chapter 6

CONCLUSION

It has been observed that, extending the naive LSH with the proposed approach shows a significant gain in terms of time taken when there are one or two excluded regions. The proposed approach avoids fetching of irrelevant candidates that are present in the excluded region. While the naive LSH fetches all the queries in the main query, the proposed approach avoids fetching irrelevant candidates in the first place. For the same parameters of LSH, naive method does have a slightly better recall compared to the proposed approach. This is due to the introduction of a few misses because of the excluded region. However, the number of candidates fetched is low compared to the naive LSH. The gain in terms of candidates is higher than the slight loss in terms of recall.

Since, none of the existing LSH algorithms inherently handle queries in the presence of excluded regions, the proposed algorithm gives a simple and effective way of doing the same. The proposed approach is an in-memory based implementation. A disk based implementation of the same would provide significant gains as well. The gain in terms of I/O costs while fetching only relevant candidates can be high compared to other disk based approaches that fetch the irrelevant candidates before filtering them.

REFERENCES

[1] "Aerial data", URL `https://www.autonlab.org/autonweb/15960.html` (2015).

[2] "Corel data", URL `https://archive.ics.uci.edu/ml/machine-learning-databases/CorelFeatures-mld/` (2015).

[3] Abel, D. J. and J. Smith, "A data structure and algorithm based on a linear key for a rectangle retrieval problem", Computer Vision, Graphics, and Image Processing **24**, 1, 1–13 (1983).

[4] Ahn, H.-K., N. Mamoulis and H. M. Wong, "A survey on multidimensional access methods", (2002).

[5] Andoni, A. and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions", in "Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on", pp. 459–468 (IEEE, 2006).

[6] Asano, T., D. Ranjan, T. Roos, E. Welzl and P. Widmayer, "Space-filling curves and their use in the design of geometric data structures", Theoretical Computer Science **181**, 1, 3–15 (1997).

[7] Bawa, M., T. Condie and P. Ganesan, "Lsh forest: self-tuning indexes for similarity search", in "Proceedings of the 14th international conference on World Wide Web", pp. 651–660 (ACM, 2005).

[8] Bayer, R. and E. McCreight, *Organization and maintenance of large ordered indexes* (Springer, 2002).

[9] Beckmann, N., H.-P. Kriegel, R. Schneider and B. Seeger, *The R*-tree: an efficient and robust access method for points and rectangles*, vol. 19 (ACM, 1990).

[10] Bentley, J. L., "Multidimensional binary search trees used for associative searching", Communications of the ACM **18**, 9, 509–517 (1975).

[11] Blott, S. and R. Weber, "A simple vector-approximation file for similarity search in high-dimensional vector spaces", ESPRIT Technical Report TR19, ca (1997).

[12] Böhm, C., S. Berchtold and D. A. Keim, "Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases", ACM Computing Surveys (CSUR) **33**, 3, 322–373 (2001).

[13] Candan, K. S. and M. L. Sapino, *Data management for multimedia retrieval* (Cambridge University Press, 2010).

[14] Casey, M. and M. Slaney, "Fast recognition of remixed music audio", in "Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on", vol. 4, pp. IV–1425 (IEEE, 2007).

[15] Casey, M. A. and M. Slaney, "Song intersection by approximate nearest neighbor search.", in "ISMIR", vol. 6, pp. 144–149 (2006).

[16] Cover, T. M. and P. E. Hart, "Nearest neighbor pattern classification", Information Theory, IEEE Transactions on **13**, 1, 21–27 (1967).

[17] Datar, M., N. Immorlica, P. Indyk and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions", in "Proceedings of the twentieth annual symposium on Computational geometry", pp. 253–262 (ACM, 2004).

[18] Datta, R., D. Joshi, J. Li and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age", ACM Computing Surveys (CSUR) **40**, 2, 5 (2008).

[19] Deerwester, S. C., S. T. Dumais, T. K. Landauer, G. W. Furnas and R. A. Harshman, "Indexing by latent semantic analysis", JAsIs **41**, 6, 391–407 (1990).

[20] Faloutsos, C., "Multiattribute hashing using gray codes", in "ACM SIGMOD Record", vol. 15, pp. 227–238 (ACM, 1986).

[21] Faloutsos, C., "Gray codes for partial match and range queries", Software Engineering, IEEE Transactions on **14**, 10, 1381–1393 (1988).

[22] Finkel, R. A. and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys", Acta informatica **4**, 1, 1–9 (1974).

[23] Fonseca, M. J., J. Jorge *et al.*, "Indexing high-dimensional data for content-based retrieval in large databases", in "Database Systems for Advanced Applications, 2003.(DASFAA 2003). Proceedings. Eighth International Conference on", pp. 267–274 (IEEE, 2003).

[24] Gan, J., J. Feng, Q. Fang and W. Ng, "Locality-sensitive hashing scheme based on dynamic collision counting", in "Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data", pp. 541–552 (ACM, 2012).

[25] Gao, J., H. Jagadish, B. C. Ooi and S. Wang, "Selective hashing: Closing the gap between radius search and k-nn search", in "Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining", pp. 349–358 (ACM, 2015).

[26] Gao, J., H. V. Jagadish, W. Lu and B. C. Ooi, "Dsh: Data sensitive hashing for high-dimensional k-nnsearch", in "Proceedings of the 2014 ACM SIGMOD international conference on Management of data", pp. 1127–1138 (ACM, 2014).

[27] Gersho, A. and R. M. Gray, *Vector quantization and signal compression*, vol. 159 (Springer Science & Business Media, 2012).

[28] Gionis, A., P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing", in "VLDB", vol. 99, pp. 518–529 (1999).

[29] Gu, X., L. Zhang, D. Zhang, Y. Zhang, J. Li and N. Bao, "Query range sensitive probability guided multi-probe locality sensitive hashing", in "Software Engineering, Artificial Intelligence, Networking and Parallel & Distributed Computing (SNPD), 2012 13th ACIS International Conference on", pp. 3–9 (IEEE, 2012).

[30] Guttman, A., *R-trees: a dynamic index structure for spatial searching*, vol. 14 (ACM, 1984).

[31] Huang, Q., J. Feng, Y. Zhang, Q. Fang and W. Ng, "Query-aware locality-sensitive hashing for approximate nearest neighbor search", Proceedings of the VLDB Endowment **9**, 1 (2015).

[32] Indyk, P. and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality", in "Proceedings of the thirtieth annual ACM symposium on Theory of computing", pp. 604–613 (ACM, 1998).

[33] Jagadish, H. V., "Linear clustering of objects with multiple attributes", in "ACM SIGMOD Record", vol. 19, pp. 332–342 (ACM, 1990).

[34] Katayama, N. and S. Satoh, "The sr-tree: An index structure for high-dimensional nearest neighbor queries", in "ACM SIGMOD Record", vol. 26, pp. 369–380 (ACM, 1997).

[35] Lawder, J. K. and P. J. King, "Using space-filling curves for multi-dimensional indexing", in "Advances in Databases", pp. 20–35 (Springer, 2000).

[36] Lin, K.-I., H. V. Jagadish and C. Faloutsos, "The tv-tree: An index structure for high-dimensional data", The VLDB Journal **3**, 4, 517–542 (1994).

[37] Liu, T., A. W. Moore, K. Yang and A. G. Gray, "An investigation of practical approximate nearest neighbor algorithms", in "Advances in neural information processing systems", pp. 825–832 (2004).

[38] Liu, Y., J. Cui, Z. Huang, H. Li and H. T. Shen, "Sk-lsh: an efficient index structure for approximate nearest neighbor search", Proceedings of the VLDB Endowment **7**, 9, 745–756 (2014).

[39] Lv, Q., W. Josephson, Z. Wang, M. Charikar and K. Li, "Multi-probe lsh: efficient indexing for high-dimensional similarity search", in "Proceedings of the 33rd international conference on Very large data bases", pp. 950–961 (VLDB Endowment, 2007).

[40] Manola, F. and J. A. Orenstein, "Toward a general spatial data model for an object-oriented dbms.", in "VLDB", pp. 328–335 (1986).

[41] Nagarkar, P. and K. S. Candan, "Hcs: Hierarchical cut selection for efficiently processing queries on data columns using hierarchical bitmap indices.", in "EDBT", pp. 271–282 (2014).

[42] Nagarkar, P., K. S. Candan and A. Bhat, "Compressed spatial hierarchical bitmap (cshb) indexes for efficiently processing spatial range query workloads", Proceedings of the VLDB Endowment **8**, 12 (2015).

[43] Nievergelt, J., H. Hinterberger and K. C. Sevcik, *The grid file: An adaptable, symmetric multi-key file structure* (Springer, 1981).

[44] Ocsa, A. and E. P. De Sousa, "An adaptive multi-level hashing structure for fast approximate similarity search", Journal of Information and Data Management **1**, 3, 359 (2010).

[45] Orenstein, J. A., "Redundancy in spatial databases", in "ACM SIGMOD Record", vol. 18, pp. 295–305 (ACM, 1989).

[46] Orenstein, J. A. and T. H. Merrett, "A class of data structures for associative searching", in "Proceedings of the 3rd ACM SIGACT-SIGMOD symposium on Principles of database systems", pp. 181–190 (ACM, 1984).

[47] Pan, J. and D. Manocha, "Bi-level locality sensitive hashing for k-nearest neighbor computation", in "Data Engineering (ICDE), 2012 IEEE 28th International Conference on", pp. 378–389 (IEEE, 2012).

[48] Rajaraman, A., J. D. Ullman, J. D. Ullman and J. D. Ullman, *Mining of massive datasets*, vol. 77 (Cambridge University Press Cambridge, 2012).

[49] Samet, H., *The design and analysis of spatial data structures*, vol. 85 (Addison-Wesley Reading, MA, 1990).

[50] Sellis, T., N. Roussopoulos and C. Faloutsos, "The r+-tree: A dynamic index for multi-dimensional objects", (1987).

[51] Slaney, M. and M. Casey, "Locality-sensitive hashing for finding nearest neighbors [lecture notes]", Signal Processing Magazine, IEEE **25**, 2, 128–131 (2008).

[52] Slaney, M., Y. Lifshits and J. He, "Optimal parameters for locality-sensitive hashing", Proceedings of the IEEE **100**, 9, 2604–2623 (2012).

[53] Sundaram, N., A. Turmukhametova, N. Satish, T. Mostak, P. Indyk, S. Madden and P. Dubey, "Streaming similarity search over one billion tweets using parallel locality-sensitive hashing", Proceedings of the VLDB Endowment **6**, 14, 1930–1941 (2013).

[54] Tao, Y., K. Yi, C. Sheng and P. Kalnis, "Quality and efficiency in high dimensional nearest neighbor search", in "Proceedings of the 2009 ACM SIGMOD International Conference on Management of data", pp. 563–576 (ACM, 2009).

[55] Tropf, H. and H. Herzog, "Multidimensional range search in dynamically balanced trees.", ANGEWANDTE INFO. , 2, 71–77 (1981).

[56] Wang, J., H. T. Shen, J. Song and J. Ji, "Hashing for similarity search: A survey", arXiv preprint arXiv:1408.2927 (2014).

[57] Wang, Q., Z. Guo, G. Liu and J. Guo, "Entropy based locality sensitive hashing", in "Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on", pp. 1045–1048 (IEEE, 2012).

[58] Weber, R., H.-J. Schek and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces", in "VLDB", vol. 98, pp. 194–205 (1998).

[59] Xie, H., Z. Chen, Y. Liu, J. Tan and L. Guo, "Data-dependent locality sensitive hashing", in "Advances in Multimedia Information Processing–PCM 2014", pp. 284–293 (Springer, 2014).

[60] Zolotarev, V. M., *One-dimensional stable distributions*, vol. 65 (American Mathematical Soc., 1986).