Efficient and Exact Range Search of Chemical Fingerprints from a Large Database

A Project Report

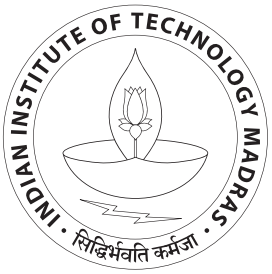
submitted by

ABHIK MONDAL

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

under the guidance of
Dr. Sayan Ranu



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

April 2015

THESIS CERTIFICATE

This is to certify that the thesis entitled **Efficient and Exact Range Search of Chemical Fingerprints from a Large Database**, submitted by **Abhik Mondal**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bona fide record of the research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Sayan Ranu
Research Guide
Assistant Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

ABSTRACT

Chem-informatics is a field dealing with chemistry and information science, with the primary motivation being the use of data mining, information retrieval and machine learning techniques to make predictions and inferences which can later be verified experimentally. The datasets used in this field are generally information regarding molecules and their structure. One such representation is in the form of ”Chemical Fingerprints”, which are primarily feature vectors .In this work we are concerned with indexing methods of such datasets of chemical compounds. We propose two techniques for the same : M-tree based technique and Inverted index based technique for performing range, point and top-k queries. We go on to study the effectiveness of these techniques through various experiments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Introduction

Chem-informatics is largely concerned with the task of mining large chemical datasets. Typical applications of this include drug discovery, catalyst analysis and experiment predictions. Usually chemicals are represented as ”Chemical Fingerprints”.Chemical fingerprints are used for identification of chemicals. They are characteristics or distinctive patterns which help describe them. The comparison of chemical molecules is difficult but if we convert the molecules into bit-strings or into vector format it makes it easier to do a comprehensive comparison. A primary reason for using chemical fingerprints is scalability. We know that sub-graph isomorphism is an NP-complete problem making it difficult to find similarity between two molecules represented in a graphical format. When the data is represented in a vector string, techniques are more scalable in terms of size of query molecules and time to accomplish a similarity detection task.

Chemical fingerprints can be built based on structure properties like substructure features. In binary fingerprints each bit represents the presence or absence of specific chemical property like substructure presence. For example the bits could represent the count of individual chemical atoms like carbon, oxygen, hydrogen or the number of saturated or unsaturated aromatic carbon only, nitrogen containing or non-aromatic rings. The reason why substructure presence is is an important feature is because it can be used to reduce or filter out candidate mismatches in the sub-graph isomorphism test process. So given two bit-strings or vectors for binary chemical fingerprints we can intuitively say that the similarity of the two fingerprints is directly proportional to the number of shared bits.

These fingerprints/feature vectors are designed by domain experts and typical features could include information like number of occurrences of particular substructures, presence of molecules, bonds between the molecules, etc. Generally the dataset is high dimensional and also highly sparse which inherently makes searching and indexing such chemical data sets challenging.

For performing query operation on high dimensional data, it is generally observed that all the indexing techniques perform poorly, as compared to linear scan. This is mainly due to the curse of dimensionality . As dimensions increases the data points are scattered further away in space. In some techniques which use pivots to form clusters like the one we will use, it is difficult to form compact groups with a low cluster radius. Since we are unable to form compact representative groups well, more than often a range query requires a search through the entire database of chemical molecules with no chance of pruning occurring. Hence, in our work we have used "Linear scan" as our base line technique and have come up with methods to improve our performance. Our experiments will be evaluated against the linear scan technique. The results of the range search will be verified by comparing the range search result set against that obtained by a linear scan approach.

Various scoring schemes like Tversky, Pearson, Dice, and Kulczynski are available for comparing two given fingerprints [**? ?**].The most widely used and accepted similarity measure for binary fingerprints is the Tanimoto similarity which is equivalent to the Jaccard Similarity index defined as the size of the intersection of the sets divided by the size of union of the sets. Here the similarity measure would correspond to the number of bits which are 1 for both the sets of the fingerprints divided by the number of bits for which atleast one of the fingerprints is 1. In mathematical terms,for binary fingerprints X and Y ,where X_i is the i^th bit of X we define Tanimoto similarity as

$$T_s(X,Y) = \frac{\sum_i X_i \wedge Y_i}{\sum_i X_i \vee Y_i}$$

Here we can show that $1 - T_s$ is a proper distance metric preserving triangle inequality which will allow us to use index structures which exploit the property, like M-trees. One challenge which we face here is the extension of the said similarity measure /distance metric to non binary fingerprints. Non binary fingerprints are basically feature vectors where each feature value is a count and not necessary signifying absence or presence using bits

Similarity between chemical molecules plays an important role in various aspects of biology and chemistry like protein ligand docking, biological activity prediction,reaction site modelling and mainly drug discovery. Chemical or molecular similarity plays a pivotal role in modern techniques used to predict chemical compound properties, designing tailor-made chemicals with a predefined and desirable set of properties and most importantly in managing drug design studies by using large databases containing

structures of available chemicals.

The first technique we used is that of M-tree [**?**] based indexing. In this technique we have proposed a novel pivot selection technique, which results in a efficient search time for range queries. We Follow this up by embedding the data in to Euclidean space using Lipschitz embedding. We then perform a detailed analysis of the data, and we show that this analysis naturally leads us to inverted indexing. The large size of the data and the sparsity of the data are the primary challenges which we have addressed in this work.

CHAPTER 2

Related Work

2.1 Related Work

In cheminfomatics the problem of fingerprint searching in a large database is well studied. In **?**] they present a new index-based search method called ChemDex (Chemical fingerprint inDexing) for speeding up the fingerprint database search. They propose a novel chain scoring scheme to calculate the Tanimoto (Jaccard) scores of the fingerprints using an early-termination strategy. The fingerprints are horizontally sorted by the total number of 1's they contain. A vertical sorting or rearrangement then takes place such that most of the 1's get pushed towards the left. After the shuffling vertical splits/slices are created in the database. A two tier index structure is created based on total number of 1's each data point has plus the number of 1's in each split. Since we have vertical slices in the database, when a query compound is given they propose a slice by slice fragment filtering.The bounds are calculated in the first slice and only the compounds which cross the threshold are checked for in the second slice. Since they are laterally traversing the slices and filtering after each slice the number of candidates are decreasing at each step hence reducing the time as compared to when we process the fingerprints in full. They come with a scoring technique such that the partial score at each slice is the upper bound for the final similarity score.

This field has also been well motivated in **?**]. As mentioned in the paper, fingerprint vectors are used to search large databases of small molecules, currently containing millions of entries, using various similarity measures, such as the Tanimoto or Tversky's measures and their variants. They derive simple bounds on these similarity measures and show how these bounds can be used to considerably reduce the subset of molecules that need to be searched.The paper proposes bounds for both single molecule as well as molecule molecule queries. The queries considered by the paper are based on threshold similarity cut-off with a query compound as well as top-k best set. The paper studied the speed-up achieved in query time against query size , query distribution, length of the chemical fingerprints, threshold cut-off for similarity and the total size of chemical database. They show through experiments that their approach achieves linear speed-ups in order of 1 or more magnitude for the fixed threshold query scenario while for top-k queries, they achieve sub-linear speed-ups in range of $O(D^{0.6})$ where D is size of database.

A different approach to the same approach can be seen in **?**] where to speed-up database searches, they proposed to add to each binary fingerprint a short signature integer vector of length M. For a given fingerprint, the i-component of the signature vector counts the number of 1-bits in the fingerprint that fall on components congruent to i modulo M. Given two signatures, they show how one can rapidly compute a bound on the Jaccard-Tanimoto similarity measure of the two corresponding fingerprints, using the intersection bound. These signatures allow one to significantly prune the search space by discarding molecules associated with unfavourable bounds.

CHAPTER 3

Problem Statement

3.1 Problem Statement

As the title suggests we are interested in fast indexing and searching of chemical fingerprints. We want to propose new indexing techniques which build on current indexing data structures and which will make searching for chemical compounds in the database faster. Our primary goal is to be able to perform queries on our compound database as fast as possible.

We are looking at accurate searching of similar compounds when given a query compound. The similarity of chemical fingerprints is established using the Min-Max distance which is the generalization of Tanimoto distance for non-binary datasets (explained in the next section). We are dealing with exact similarity match and not approximate similarity . We will be looking at different types of queries, namely the range, point and top-k queries .

Range queries are of the form , where given a fingerprint, say *f*, and a threshold similarity θ we want to find the set S of all fingerprints, such that

$$sim(f, g) \ > \ \theta \ \ \Rightarrow g \in S$$

Top k search queries would require us to find the top k most similar compounds to the query compound. Given a fingerprint, say f and a value *k*,we want to find the Set S of size k such that

$$sim(f, g) \geq sim(f, h) \ \ \forall g \in S, \ \forall h \notin S$$

. We haven't looked at these search queries in the experiments as of yet but plan to explore this area in the future.

For the inverted indexing technique we will be looking at point queries as a start and in the future try to come up with a better technique for range queries and top-k queries.

Our aim as mentioned earlier is to come up with a novel indexing scheme which would make the aforementioned tasks faster. For range queries we would want to achieve sub-linear search time speeds.The challenge and novelty in our work comes from the fact that, unlike in **?**] and many other state of the art techniques developed in this domain, we are working on non-binary vector fingerprints.

CHAPTER 4

Similarity Measure

4.1 Similarity measure

As mentioned in the earlier section the main challenge is the fact that we are dealing with non binary fingerprints which has seen very little work. Many of the papers mentioned in the related work suggest that their technique can be extended to non-binary fingerprints but fail to provide any experimental evaluation nor any details about the similarity measure used. We will be using a generalization of the Tanimoto similarity measure to incorporate non binary feature values. The similarity measure known as Min-Max similarity measure M_s between two fingerprints X, Y can be formulated as, where X_i is the i_{th} feature value of X

$$M_s(X,Y) = \frac{\sum_i min(X_i, Y_i)}{\sum_i max(X_i, Y_i)}$$

If the fingerprints are binary, this similarity measure reduces to the Tanimoto similarity. We need to ensure that the corresponding distance measure $M_d = 1 - M_s$is a metric. We can write the distance measure as:

$$M_d(X,Y) = 1 - M_s(X,Y) = 1 - \frac{\sum_i min(X_i, Y_i)}{\sum_i max(X_i, Y_i)}$$

$$M_d(X,Y) = \frac{\sum_i |X_i - Y_i|}{\sum_i max(X_i, Y_i)}$$

We can easily see that $M_d(X,X) = 0$ and that if $M_d(X,Y) = 0$ it implies $X_i = Y_i$ for all i, hence implying X=Y. The triangle inequality has been proved in **?**]

CHAPTER 5

M-tree Based Indexing

5.1 M-tree Based indexing

5.1.1 M-tree Data Structure

M-tree also known as the Metric tree is a tree data structure constructed using a metric distance measure and relies on the triangle inequality for efficient range search queries. Similar to all other tree data structures, an M-tree data structure also has Leaf Nodes and non- Leaf nodes. Every non leaf node has a pointer to its parent node, a pointer to its subtree, its object information and a covering radius denoting maximum distance of a node to any node in its subtree.Every leaf node keeps a pointer to its parent and object information.

The M-tree data structure compartments the objects into nodes, which define regions of the metric space. The maximum capacity of the M-tree is M. For each database object to be indexed, there is an entry $O_j = [O_j, id(O_j), dist(O_j, P(O_j))]$ in some leaf node. O_j stores the object information, mainly the data point feature values or dimension coordinates, id(O_j) stores the id of the object and $dist(O_j, P(O_j))$ stores the distance of it to its parent object.

A non-leaf node O_r stores an object information as well as a covering radius $r(O_r)$ and a pointer to a subtree i.e. entry $O_r = [O_r, ptr(T(O_r)), r(O_r), d(O_r, P(O_r))]$. One property satisfied is that every object O_j stored in a subtree rooted at router object O_r is at atmost $r(O_r)$ distance to it $d(O_j, O_r)r(O_r)$. M-tree organizes the metric space into a set of, possibly overlapping, regions, to which the same principle is recursively applied.

5.1.2 Information in the data structure

We use a modified data structure similar to the M-tree.We keep the following information in each node of the our modified M-tree: the object identifier of the pivot, i.e fingerprint information of the pivot fingerprint; the pivot also stores a pointer to its children i.e a pointer to a set of fingerprints; a double value which is the distance of the farthest child in its subtree and a boolean value which signifies if the pivot is an outlier pivot. We do not require a parent pointer. And the size of each node is determined by the number of pivots chosen at each step of our indexing technique and the base outlier size limit, which are explained later. The following information is stored in each entry of a node in our tree:

1. Object identifier p_i
2. Pointer to subtree S_i
3. Farthest child i.e. Covering radius r_i
4. Outlier pivot boolean variable

5.1.3 Baseline Indexing approach

In the baseline approach, we are partitioning the dataset into groups using pivots which enables us to exploit the triangle inequality. The choice of pivots in the baseline approach is done randomly. The number of pivots is determined by two input parameters viz. the number of random pivots chosen at each step and the minimum size of the outlier set allowed at the lowest level. This has been formally explained in the algorithm below.

1. Choose the given number of random pivots from the set. The number of random pivots chosen at this step is a parameter to our experiment and has been varied across different values.
2. After choosing pivots we assign every other chemical compound in the database to one of the pivots based on the similarity to the pivots. A chemical compound is assigned to the pivot which is nearest in distance to it i.e. it has the highest Min-Max similarity with that as compared to other pivots.
3. The next step involves ranking all chemical compounds data points by their similarity to their own closest pivot.
4. We calculate the median similarity among all the chemical compounds in this database and note it as the outlier cut-off/
5. All the chemical compounds with similarity less than the cut-off similarity is called as outliers in this step

6. The outliers are unassigned from the pivots and assigned to the ”outlier” pivot
7. We recursively apply this technique on the outlier set until the outlier set size reaches a base limit which is also a parameter to our algorithm.

We tried an alternate approach where instead of choosing outliers after assigning molecules the most similar pivot we recursively apply the technique on each set. In simpler words, we try to cluster at each step and try to create clusters among each cluster recursively. So in each cluster we choose pivots again and try to assign each of the points to the pivots until we cannot choose more pivots. We can formally explain it as follows.

1. Choose the given number of random pivots from the set. The number of random pivots chosen at this step is again a parameter to the experiment as previously.
2. The second step is also similar to the earlier indexing technique. After choosing pivots we assign every other chemical compound in the database to one of the pivots based on the similarity to the pivots.
3. Instead of choosing outliers now, we work with the same sets and recursively apply this technique on the set until the set size reaches a base limit .
4. The set size limit is set as the number of pivots chosen at each step i.e. the method is applied recursively on the set until the size of the set is lower than the pivot set size making us unable to choose pivots.

The previous indexing technique was seen to give us better results. Hence we will be using the earlier technique for all experimental evaluations. One reason we could think of why this technique failed is probably because the tree was becoming more imbalanced and the height was increasing causing querying time to be more.

5.1.4 Choosing pivot

Instead of randomly choosing pivots at every step we use a Max-Min distance approach where we try to maximize the minimum distance between any two pivot nodes chosen. The procedure can be explained as below:

1. Choose a random point.
2. The second point chosen is such that it is farthest from the first chosen random point. This will require a full database scan

3. The third point is chosen such that its minimum distance to either of the previous two pivots is maximized.
4. Iteratively, the i^{th} point is chosen in a fashion such that the minimum of its distance to the previous i-1 points is maximized.
5. This procedure is repeated till we choose p points where p is the number of pivots to be chosen

We can observe that this is a very expensive procedure since we are scanning the entire database at every step. We need to compute an all pair similarity initially. At the i^{th} step , the remaining n-i+1 points in database need to compute its minimum distance to the i-1 points already present in pivot step. Finally choose the point among the n-i+1 points with the maximum distance. This computation is of the O(n^3).

$$T(n) = n^2 + \sum_{i=1}^{n}((n-i+1)(i-1) + n - i + 1)$$

$$T(n) = O(n^3)$$

We use a sampling technique where we randomly sample a subset of the database and apply the technique described above on the subset. We used a subset size of $\frac{1}{10^{th}}$ the size of the database, approximately about 20,000 points in the subset.

5.1.5 Range Search Querying

Given a query chemical fingerprint q and a threshold θ we want to find the set of chemical fingerprints which satisfy the query. We exploit the triangle inequality for the same. The procedure for range search querying can be described by the following steps

1. The basic idea is to be able to prune subtrees based on the covering radius of the pivot of the subtree and the distance of the query to the pivot
2. Let the query fingerprint be q and the fingerprint pivot be p_i of subtree S_i. We can calculate the maximum distance of any node in S_i from q.We start with the root of the tree as p_i
3. Let the covering radius of pivot p_i be r_i. Hence the maximum distance of any node in S_i will be $dist(q, p_i)$ + r_i. Similarly the minimum distance of any node in S_i is $max(dist(q, p_i)$ - $r_i, 0)$.
4. Hence we can calculate the range of the distance of any node in S_i to q.
5. If the upper bound of the range or the maximum distance is lesser than the threshold distance θ we can add all the nodes in S_i to our resultant set

6. If the lower bound of the range is greater than the threshold distance θ, we can prune the subtree S_i since we can say with certainty that the distance of every node in the subtree S_i is greater than the threshold θ.
7. If there is an intersection in the intervals we recursively apply this technique on the second level of children in the subtree S_i until we reach a leaf node.

5.1.6 Lipschitz Embedding

The performance of the M-tree is limited by the fact that the data is very sparse. This results in very loose clusters being formed during the indexing phase. To improve upon this we performed an embedding of the data into Euclidean space using Lipschitz embedding. Lipshcitz embedding results in a contractive mapping.

Lipschitz embedding is the embedding, or in simpler words, dimensionality reduction of the objects of a database D with metric distance d onto a k-dimensional feature vector space. The procedure goes as follows:

1. Choose k subsets of D
2. Each subset A_i is a reference set
3. Let the distance of object p_j to set A_i be:
$$d(p_j, A_i) = min_{a \epsilon A_i} d(p_j, a)$$
4. The feature vector f(o) is then defined as:
$$f(p_j) = \frac{d(p_j, A_1)}{k}, \frac{d(p_j, A_2)}{k} ... \frac{d(p_j, A_k)}{k}$$
5. We define the new distance $d'(f(p_j), f(p_k))$ as :
$$d'(f(p_j), f(p_k)) = \sum_{i=1}^{k} |d(p_j, A_i) - d(p_k, A_i)|$$

We chose the reference sets in our embedding procedure as singleton sets. The choice of these points is done such that they are maximally far apart, i.e., the average and minimum all pairs distance among the reference sets is maximized similar to the method of choosing pivots in our indexing technique. This criteria of selecting the sets was based on the heuristic that farthest point will be maximally representative of the dataset. Since choice of such an optimal set is NP-hard, we perform a randomized set construction.

1. Choose a random pivot from a sampled set of data set points
2. Add the point to our Reference set
3. Find the point farthest from the reference set from the sampled set of points
4. Add the point to the reference set
5. Repeat step 3-4 until Reference set is full

The dimension of the embedded space was chosen to be 514, which is the square root of the total number of points in the data set.We then performed M-tree based indexing and searching on this dataset. Our performance had deteriorated from that of the baseline technique. In later section we shall give a reasoning for this deterioration.

CHAPTER 6

Experiments - M-tree Indexing

6.1 Experiments - M-tree indexing

For experiments , the dataset we are using is the size of 200000 chemical compound fingerprints with 500 random query fingerprints chosen from the total set of about 250000 fingerprints. For the experiments the test bed used was a 4 Intel(R) Core(TM) i7-4770 CPU 3.40GHz with 8GB RAM. We have varied several parameters in the experiment .For evaluation purposes we implemented a linear brute force scan to compute the range query and used that as a benchmark for the results. The result set obtained from our technique was also compared with the linear brute scan answer set for verification purposes using the fingerprint id's. The query time and the indexing time is averaged over the 500 query sample data points and the unit in ms per compound. We have varied the following parameters:

1. No. of random pivots chosen at each step.
2. Minimum size of the outlier set allowed (When to stop the indexing)
3. The range query distance threshold.

6.1.1 Different pivots

We observe that indexing time is increasing almost linearly with the number of pivots chosen at each step. We see that the number of comparisons needed to be made is decreasing with increasing pivots till 2000 for a dataset of about 200000 beyond while indexing time increases significantly We also observe that query time is also similarly reducing with increase in pivots till 2000.

6.1.2 Different outlier base size

We observed that the outlier base size did not have a significant effect on the query time for range search or on number of comparisons. The indexing time increases with a lesser outlier base size because the depth of the M-tree increases . The algorithm is applied recursively on the outlier sets, hence when we

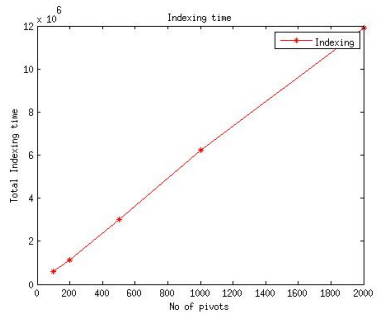


Figure 6.1: Indexing time (in ms): As seen in the figure indexing time is increasing almost linearly with increase in the number of pivots chosen at each step of the M-tree algorithm. Even though indexing is an offline process we do not want the indexing time to run into days. If indexing time is very high, updates to the chemical compound database would be very expensive since we need a red-indexing into the chemical database. Hence we need to have a threshold for indexing time. The threshold distance used for this experiment was 0.3

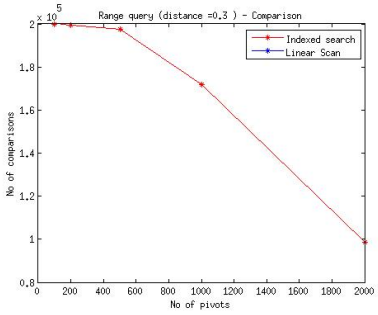


Figure 6.2: Comparisons: As seen in the figure, the number of comparisons i.e. the number of leaves containing chemical compounds which are visited in the M-tree index can be seen to decrease with increase in the number of pivots. We can notice that for number of pivots chosen as 2000, we are able to prune away more than half the nodes in the M-tree . The threshold distance used for this experiment was 0.3

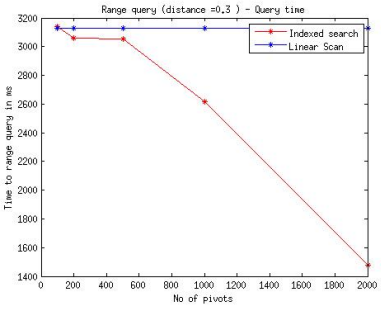


Figure 6.3: Query time (in ms): As seen in the figure, the query time can be seen to decrease with increase in the number of pivots. For pivot set size of 2000 we are able to achieve more than 2-times speedup. The linear scan be seen to take about 3100 ms per compound our technique can be seen to take around 1500ms per second for the specific size of 2000 pivots.The threshold distance used for this experiment was 0.3

have a lower base size limit for the outlier sets the number of times the recursion is applied is greater. Since we did not see a great change in query time or number of comparisons we have fixed the size the outlier size limit to 100 i.e the algorithm terminates when the outlier size becomes lesser than 100.

6.1.3 Different distance range queries

With increasing range distance query , we can see that both the no of comparisons and query time increases with the distance. This is for a 100,000 data point set. The widely accepted similarity threshold cutoff using the Tanimoto distance is 0.3 which was used in the previous experiments. For lesser distance range queries we get a speed-up of more than 10 times. We have varied the threshold from 0.05 to 0.5 in steps of 0.05. It doesn't make sense to go beyond 0.5 since queries generally want to find similar compounds and 0.5 seems like the maximum threshold. We should expect the curve to fall beyond 0.5. This is because using the triangle inequality bounds, for higher values of threshold distacne we should be able include many subtrees in our answer without having to compare distance of query to all these points.

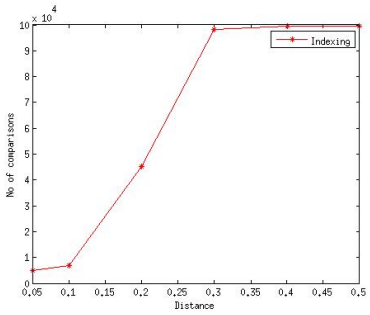


Figure 6.4: Comparisons: The figure shows the number of comparisons made when different threshold values are used as similarity cutoff. As noticed in the figure, for lower values of threshold , pruning can be applied to more than 90 percent of the nodes . As the threshold reaches 0.5 very little pruning occurs.

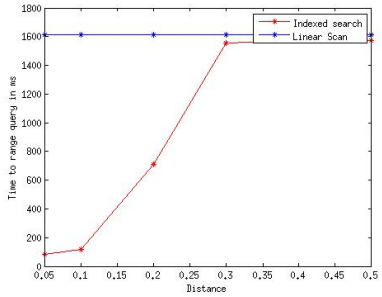


Figure 6.5: Query time (in ms): The figure shows the query time when different threshold values are used as similarity cutoff. As can be seen in the figure, for lower values of threshold almost 15 times speedup can be achieved though for threshold values close to 0.5 the query time increases and becomes comparable to the linear scan query time.

CHAPTER 7

Inverted Indexing

7.1 Inverted indexing

7.1.1 Analysis of the Data

We performed a detailed analysis of the data to figure out the kind of indexing technique, which needs to be used to optimize our searching time. The statistics that were extracted from the data are as follows:

Number of data points	264016
Number of unique features is	785985
Maximum number of features in a data point is	1903
Minimum number of features in a data point is	7
Average number of features in a data point is	270.602966
Maximum number of data point with a feature is	259110
Minimum number of data point with a feature is	1
Average number of data point with a feature is	90
Maximum value of a feature	1870
Minimum value of a feature	1
Average value of a feature	1.142210
Maximum number of heavy-hitters	144
Minimum number of heavy-hitters	1
Average number of heavy-hitters	44.5

Table 7.1: Statistics of Data

From this we can observe that the data is highly sparse with only about 271 features, on a average, in a point, as opposed to the 785985 unique features. This high sparsity makes the data set an ideal candidate to perform inverted indexing. In inverted indexing, instead of indexing the data points we would index the features. This leads to a large index structure, but we gain on the speed of point query.

7.1.2 Inverted indexing

We construct the index structure on features. Our index structure is a chain hashing structure. For a give data point to be inserted in to the structure, for each of it features, we hash the feature and insert the reference to the data point in the chain. In our current implementation we have used a hash table of size 785985, and hence we do not have any collisions. The hash table is maintained as a binary search tree. This structure results in a chain of maximum length 264016.

Point Query

Point query operation can be very effectively implemented in the inverted index structure. To perform this operation we maintain a list of candidate nodes, which needs to be explored sequentially. Our aim is to prune the candidate list as much as possible. This pruning is done by the following observation. Given a point p we look at all its features, and find the list of all nodes associated with the features. It is a straight forward observation that p is present in the dataset only if, it is in the intersection of all the associated point list. This reduces our candidate list to a size, atleast as small as the smallest list, among all the features present in p (which in our case is on an average only 271).

Range Query and K-nn

Range query and K-nn are much more complicated queries, when compared to a point query. The candidate set in this case is much more diverse and distributed. For example the candidate set for a Range query will be the union of all the point lists associated with the features of p, which is the worst case is of the order of N. We try to prune this by observing that the minimum number of features in a point is 7, hence we can remove the top 6 features from the index with out affecting the accuracy of searching. But this pruning does not result in any improvement in performance. We shall discuss this further the sections below.

CHAPTER 8

Experiments - Inverted Indexing

8.1 Experiments - Inverted Indexing

The results of point query experiments are as follows

Data size	Avg points explored		Avg Time (sec)	
	Inverted	linear	Inverted	linear
1000	6.87	727.6	0.02	0.764
10000	54.57	7277.67	0.1	9.76
100000	334.37	73008.21	0.5	83.09
FULL	1972.75	83886.07	2.32	225.68

Table 8.1: Point query

As can be seen our technique is able to achieve up to 100 fold improvement over the linear scan. This was made possible by the pruning that was possible on this dataset. Column 2 and 3 give us a insight into the amount of pruning that was achieved by this technique. We are able to achieve up to 80 times more pruning.

CHAPTER 9

Data Characteristics

9.1 Affect of Data on our Techniques

To get some more insight in to the behaviour of the indexing techniques that we applied, we performed further more analysis of the data. This gave us some further useful explanations.

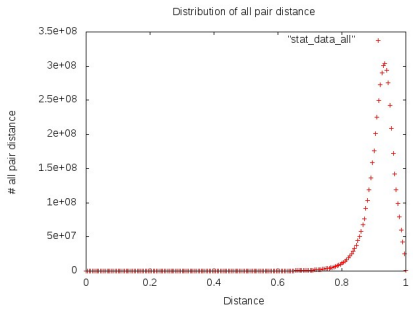


Figure 9.1: All pair distances

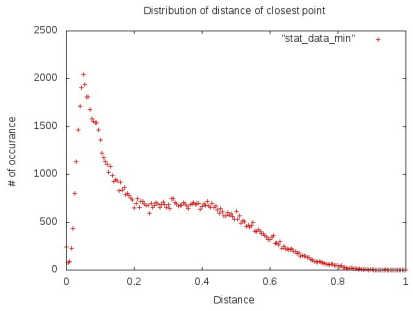


Figure 9.2: Distance of the closest point

What we can observe from the data are the following:

1. The being in high dimension, is very much spread out, and hence most of the points are equidistant from each other.

2. The closest point for most of the points is at distance much greater than 0.4. in fact only 0.007% of the points have their 1-NN within a distance of 0.4.

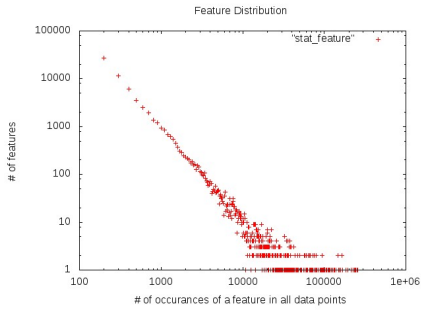


Figure 9.3: Feature distribution

3. the distribution of features among the data points seem to follow a power law distribution (though we have not tried to regress the plot to a function) i.e., even though we have many features only a hand full of them are repeated in most of the points.

These observations have many profound impact on our algorithms.

A. One of the most time consuming operations in our algorithms has been the recurring theme of finding the set of maximally separated points, which we called as pivots. It takes hours for the algorithm to converge. Given the distribution of all pairs distance, we cannot hope to improve it further, but we can propose an heuristic streaming algorithm to handle it efficiently.
We can maintain a list of most frequently occurring farthest points. These points are not necessarily the points farthest from each other, but the points which are most frequent in the list farthest points, for each point. This is motivated from the frequency counting problem from streaming as shown in **?**].

B. We observed that the inverted index we proposed could not be easily generalised to range queries. This can be further observed from the fact that how skewed the data distribution is.
Given that only 0.007% of the 1-nn points have distance less that 0.4, we can never achieve a efficient pruning. And in addition the distribution of heavy-hitters (features with occurrence more that 50000) is also very high, We have on an average 44 heavy hitters in a data point. Hence, in the worst case, we will be forced to explore all the points.

C. The success of any embedding technique, especially Lipschitz, depends on the ability of the reference set to be representative of the entire data. in our case, given that the entire data is very widely spread, the number of reference set required to well represent the data, becomes very large. This is the reason why Lipschitz did not give the desired results.

CHAPTER 10

Conclusion

10.1 Conclusions and Future Directions

In this work we were able to show the effectiveness of M-tree based approach on non-binary feature vectors. Though we could achieve up to 2-fold improvement in query time, we would want to improve this further by doing further analysis.We also were able to effectively exploit the sparsity in data to construct the inverted index, which had given up to 100-fold improvement in query time.The current work has opened up many more avenues for us to explore. We shall be looking into the following areas:

1. Use of streaming algorithm as mentioned in the previous section.
2. We also observe that we can replace Binary Search Tree in the inverted index technique by a height balanced data structure to result in even more efficient index structure.
3. One other direction which is a worthy endeavour is the exploitation of the fact that the maximum value taken by any feature is bounded. We may be able to come up with some good bounds on the distance metric based on this, these bound can be in turn used to provide good pruning.
4. We will also be looking into other dimensionality reduction techniques like principal component analysis, fast map, etc. Since we are looking into exact range search queries, we must be able to give guarantees on the new mapping, preferably a contractive mapping which will be a challenge

CHAPTER 11

Future Work

REFERENCES