Indexing chemical fingerprints for efficient querying of molecular databases

A Project Report

submitted by

ABHIK MONDAL

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

BACHELOR OF TECHNOLOGY
&
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 **Indexing chemical fingerprints for efficient querying of molecular databases**, submitted by **Abhik Mondal**, to the Indian Institute of Technology, Madras, for the award of the degrees of **Bachelor of Technology** & **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. Chemical Compounds are frequently represented as feature vectors where every feature represents the absence, presence or the frequency count of certain important substructures or other chemical properties. These feature vectors are known as ”Chemical Fingerprints” .

Fast database search is vital especially in drug discovery where the aim is identifying chemical compounds with high similarity to known drugs.In this work we are concerned with indexing methods of such datasets of chemical compounds to facilitate rapid range search querying on the database. We propose two techniques for the same. There is no loss of accuracy in any of the two methods when compared to a complete database linear scan .

The first technique uses an indexing technique based on the structure of the M-tree data structure. The standard similarity measure used for chemical fingerprints is the Tanimoto Similarity which satisfies triangle inequality. The M-tree structure helps us exploit this fact. The second technique we describe is based on an Inverted Indexing method which takes advantage of the sparsity and high dimensionality of a typical chemical fingerprint dataset. 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

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 [12, 13].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

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

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 [6] 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

In cheminfomatics the problem of fingerprint searching in a large database is well studied. In Aung and Ng [2] 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 Swamidass and Baldi [12]. 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 Nasr, Hirschberg, and Baldi [11] 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 Formulation

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.

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 Swamidass and Baldi [12] and many other state of the art techniques developed in this domain, we are working on non-binary vector fingerprints.

3.2 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

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

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:

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

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

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 Lipkus [9]

CHAPTER 4

M-tree Index 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.

4.1 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

4.2 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.

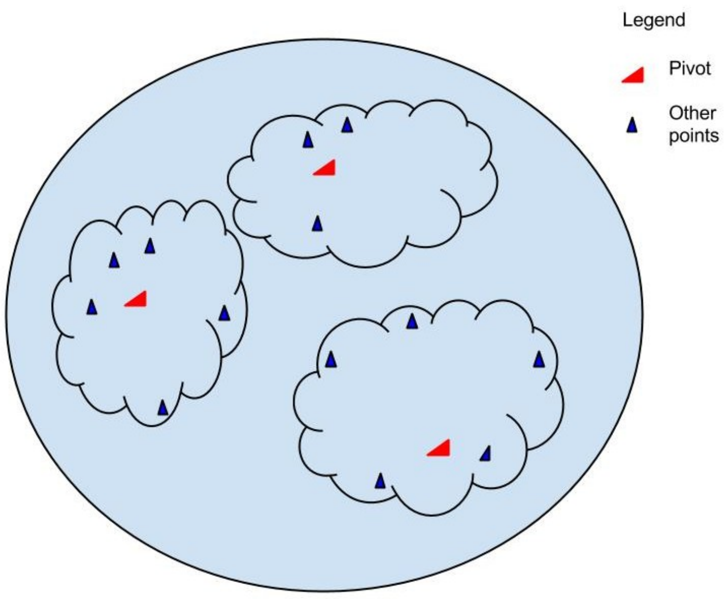

Figure 4.1: M-tree: Assigning points to a cluster after the pivoting step of the M-tree indexing process

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.

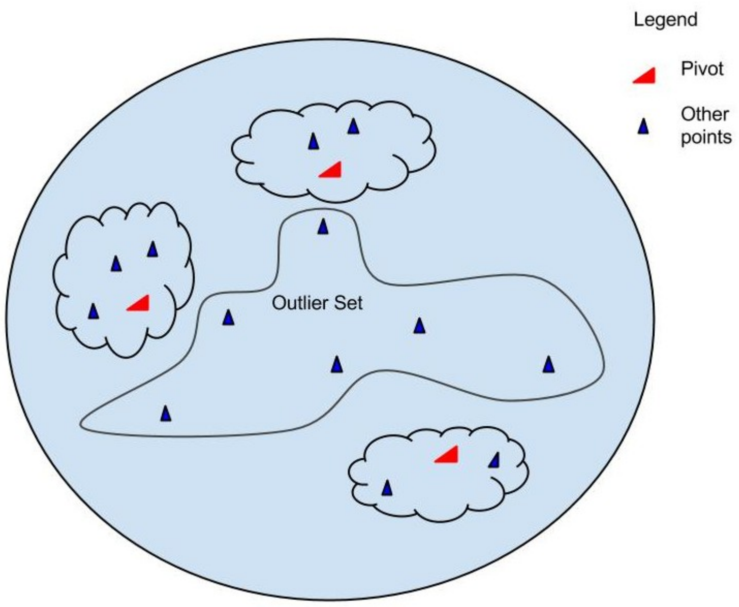

Figure 4.2: M-tree: Finding the outlier set in the M-tree index structure

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 . This is described in Figure 4.1.
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. This has been described in Figure 4.2
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.

4.3 Alternate indexing approach

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.

4.4 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.

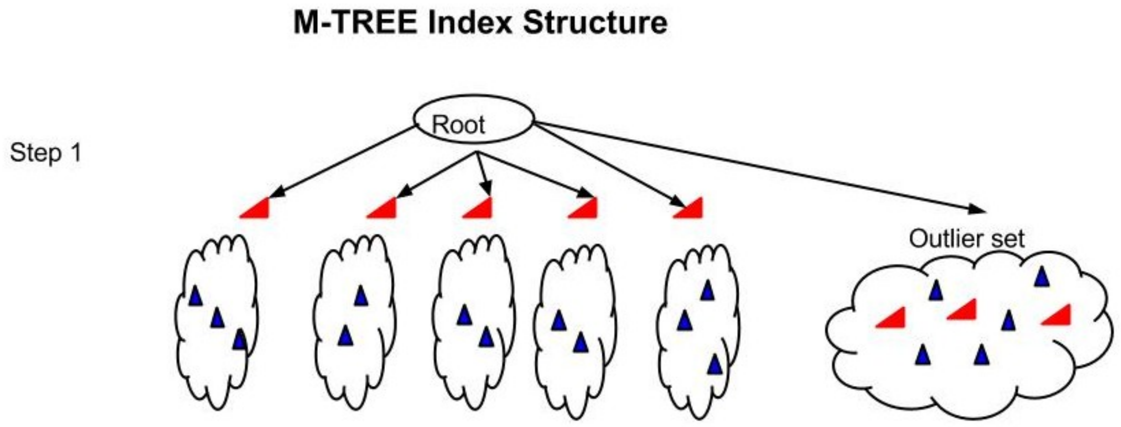

Figure 4.3: M-tree: Tree structure Step 1

4.5 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

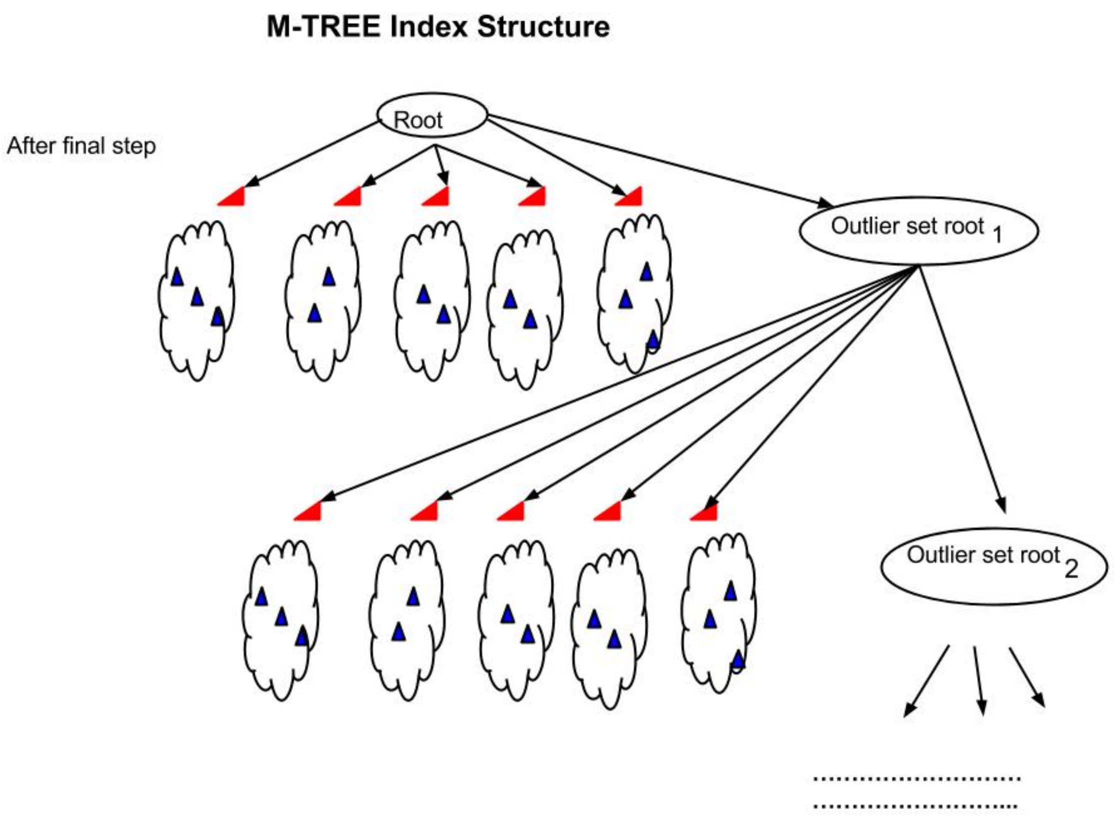

Figure 4.4: M-tree: Tree structure - After final step

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.

4.6 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 for the Swamidass dataset 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 5

Inverted Index Structure

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.

5.1 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).

5.2 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.

5.3 Baseline technique

Given a query q, which has features $f_1, f_2, ...f_{N_q}$ we take a union of all compounds present in atleast one of these features and then proceed with a linear search. This is described in detail below.

1. We keep a hash table where every feature has a pointer to the set of points which contains that feature itself.
2. Let the feature f_1 of query q be present in the points $p_1, p_2, ...p_M$. Let this set be S_1.
3. We take a union of all such sets $S_1, S_2, ...S_{N_q}$ defined as the set S_q
4. We do a linear search on the compounds present in this set.
5. The answer are the subset of points of the set S_q which fall within threshold t of the query.

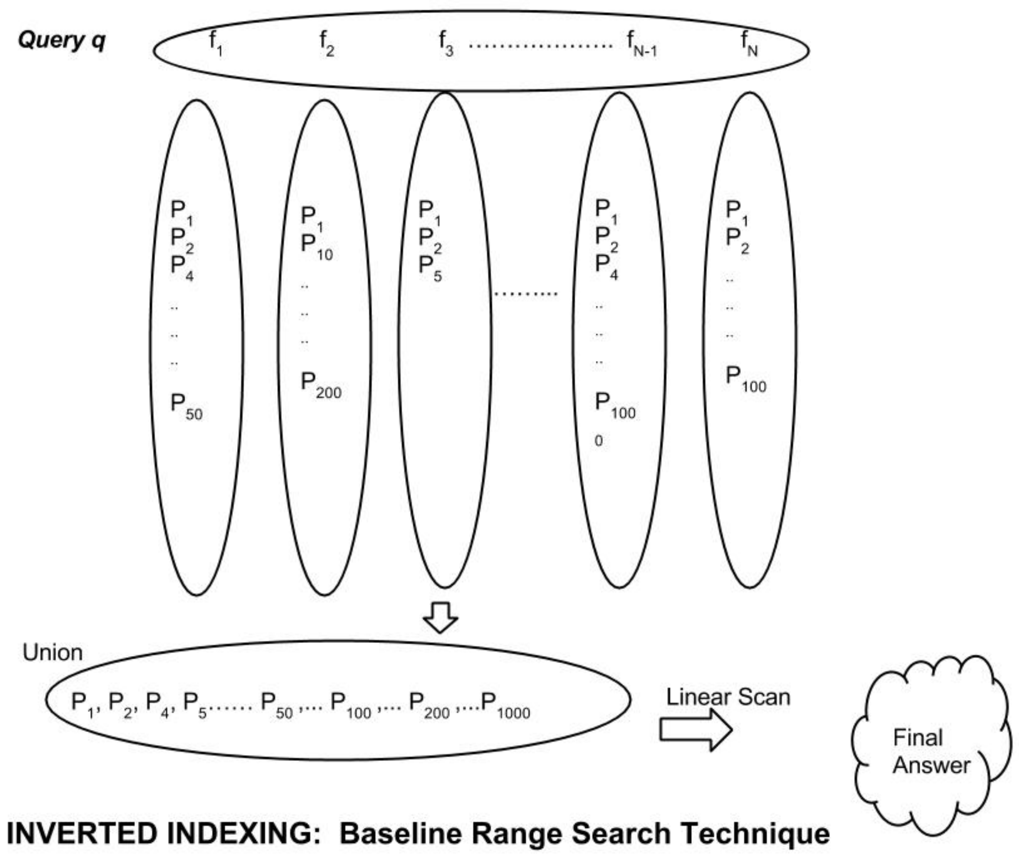
Query q
f1 f2 f3 .... fN-1 fN
P1 P2 P4 .. .. .. P50
P1 P10 .. .. .. P200
P1 P2 P5
P1 P2 P4 .. .. .. P1000
P1 P2 .. .. .. P100
Union
P1, P2, P4, P5...... P50 ,... P100 ,... P200 ,...P1000
Linear Scan
Final Answer
INVERTED INDEXING: Baseline Range Search Technique

Figure 5.1: Inverted Indexing: Procedure Diagram

5.4 Pruning Features

We use a greedy technique to prune features i.e. prune out features whose points are guaranteed to lie outside distance threshold.

1. We sort the features of the query q, $f_1, f_2, ...f_{N_q}$ in descending order of popularity i.e. the feature present in most of the points is at the start.

2. We prune this feature if the maximum similarity value is less than the minimum required similarity threshold.

3. Consider feature f_1. Let's say we want to prune this feature. We are concerned with points having this feature but not having any of the other features of the query point.

4. Let us assume binary fingerprints for simplicity. The maximum similarity such a point can have with the query point is $\frac{1}{N_q - 1 + V_{f_1}}$ where N_q is the number of features in query q and V_{f_1} is the minimum number of features present among all points having the feature f_1

5. Hence if distance threshold is t and the following holds

   $$\frac{1}{N_q - 1 + V_{f_1}} < 1 - t$$

   we can prune the feature f_1

6. We continue a greedy approach, where we consider the next feature f_2 and try to prune both the features f_1 and f_2 together.

7. Hence if till the i^{th} feature is considered, if j features have been pruned, we can prune the i^{th} feature as well if the following holds

   $$\frac{j+1}{N_q - 1 + min(V_{f_i}, minimum\ set\ J\ value)} < 1 - t$$

   where *minimum set J value* is the minimum number of features present in any of the compounds of the features pruned till now.

8. We can extend this to non binary fingerprints as well.

CHAPTER 6

Data

Before we get to the experiments, we will describe the data in this chapter. To get some more insight in to the behaviour of the indexing techniques that we are applying, we perform further analysis of the dat in the hope of getting some useful explanations. We want to ascertain that a typical fingerprint dataset is high dimensional and highly sparse almost follwing a power law sort of distribution with very few features occuring in almost all the points in the chemical database while majority of the features have a non-zero value in very few points. We want to extract patterns in the data which could be exploited for range search querying and point querying.

6.1 A Typical Fingerprint Dataset Analysis

We perform a detailed analysis of the data from **Dataset 1** 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 6.1: 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.

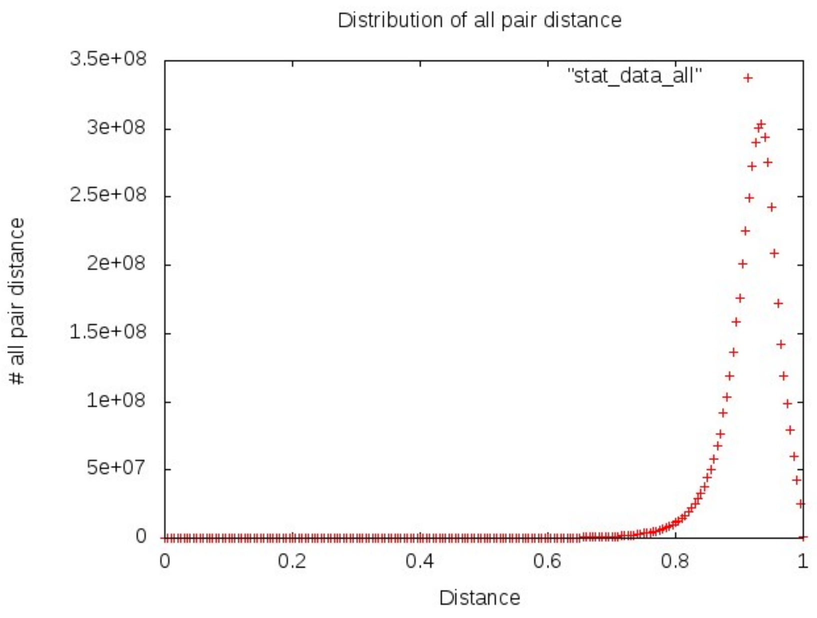

Figure 6.1: All pair distances

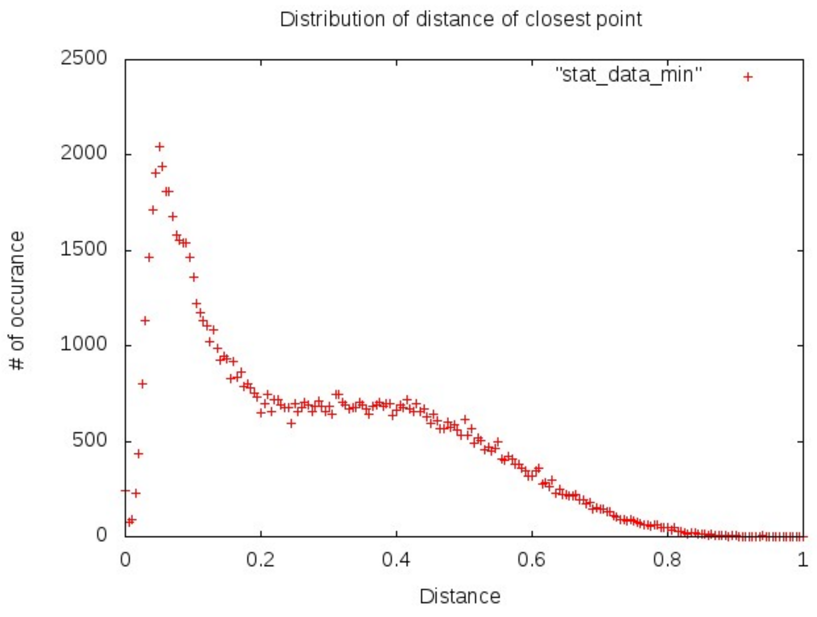

Figure 6.2: Distance of the closest point

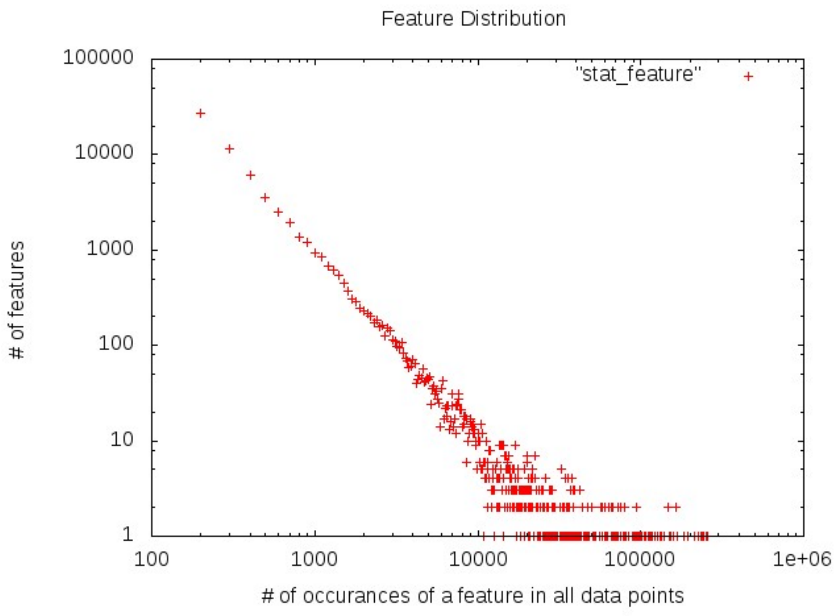

Figure 6.3: Feature distribution

6.2 Observations on the Data

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.

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.

4. 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 Metwally, Agrawal, and El Abbadi [10].

5. 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.

6. 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 7

Experiments

In our experiments, we evaluated our indexing techniques on two real world datasets. We have compared our indexing techniques by comparing the running time with that of the state of the art technique . We are also concerned about the indexing time, especially for the M-tree index structure since we do not want our index procedure to run into hours.

The first dataset is the Swamidass dataset . The second dataset is from DUD, a directory of useful decoys for benchmarking virtual screening. DUD is provided by the Shoichet Laboratory in the Department of Pharmaceutical Chemistry at the University of California, San Francisco (UCSF). DUD is derived from ZINC, a database of commercially avaiilable compounds. To extarct fingerprints we used MOLPRINT2D, a molecular fingerprinting technique.

7.1 M-tree based Indexing analysis

For these set of 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 and have tried to estimate them emperically. We have used the non-binary version of the Swamidass dataset for all analysis of the M-tree. We use different dataset sizes of 1000, 10,000 and 100,000 number of chemical compounds.

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 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 random query sample data points and the unit in ms per compound. We have varied the following parameters .

7.1.1 Limiting Outlier Set Size

Just to recap, it is the Minimum limiting size of the outlier set allowed (o). When the outlier set size falls below this limit, we terminate the indexing process.

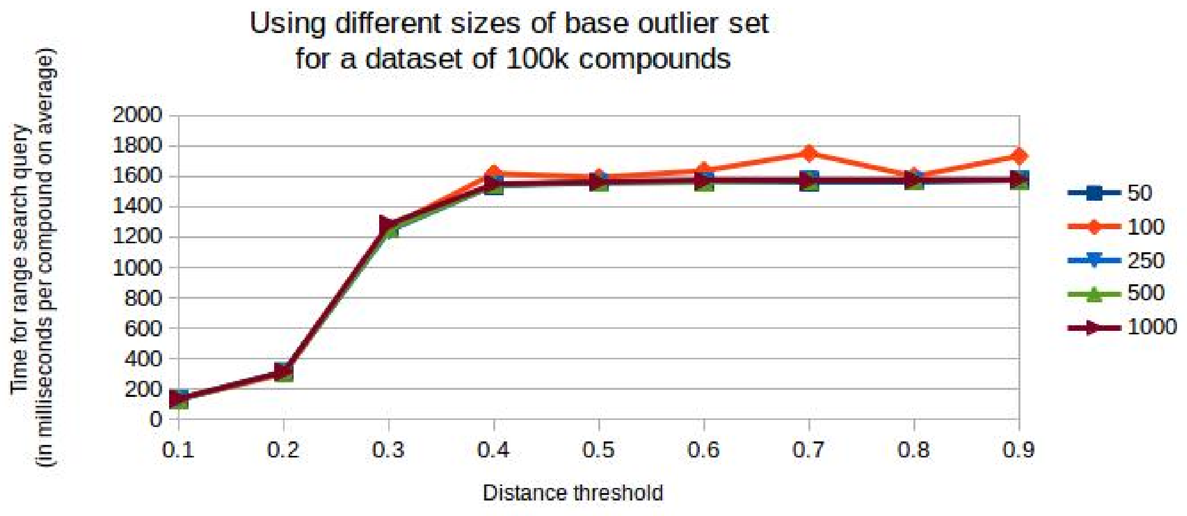

Figure 7.1: M-tree: Average Range Query time versus Distance Threshold for various sizes of base outlier set

As seen in 7.1 we can observe that the range search query time is almost constant with change in number of limiting outlier set size. We observed that the outlier base size did not have a significant effect on the query time for range search or on the 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 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 1/100th of the dataset size for all the future experiments.

Ideally, indexing time should decrease with increase in number of limiting outlier size. This is because the depth of the M-tree grows with increase in the limitng size of the outlier set. The indexing process is a recursive procedure and termination occurs only when the size of the outlier set falls below the given limiting size. Hence in normal circumstances the indexing time must decrease as we increase the limiting size, but as seen in Figure 7.2 the decrease is minimal and changing o doesn't change the indexing time much.

7.1.2 Pivot Set size

The pivot set size as described earlier is the number of random pivots chosen at each step of the indexing process (p). p determines the number of nodes at every level of the index structure. The number of nodes

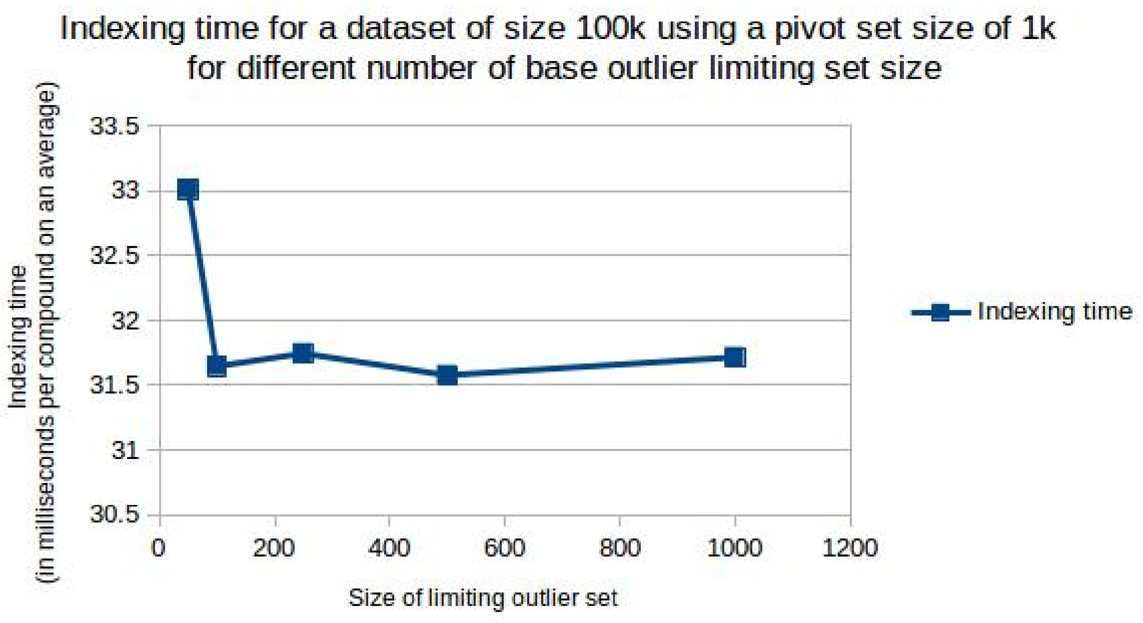

Figure 7.2: M-tree: Indexing time versus different sizes of base limiting outlier set

at each level is equal to p+1.

Figure 7.3,Figure 7.4,Figure 7.5 show how the range search query time varies versus different distance threshold for different sizes of the pivot set for databases of size 1,000 , 10,000 and 100,000 compounds respectively. We can observe the time is almost similar at high threshold values for different number of pivots. For low theshold values we are succesfully able to prune away all many nodes using triangle inequality.

We observe that 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 re-indexing into the chemical database. Hence we need to have a threshold for indexing time.

7.1.3 Threshold Distance

As described in earlier sections, the range query distance threshold (t) is thse cutoff used for determining similarity. We observe that range search query time increases monotonically with threshold distance and begins to converge for higher values of threshold distance. But this seems to hold true only if we use pivots above a particular threshold.

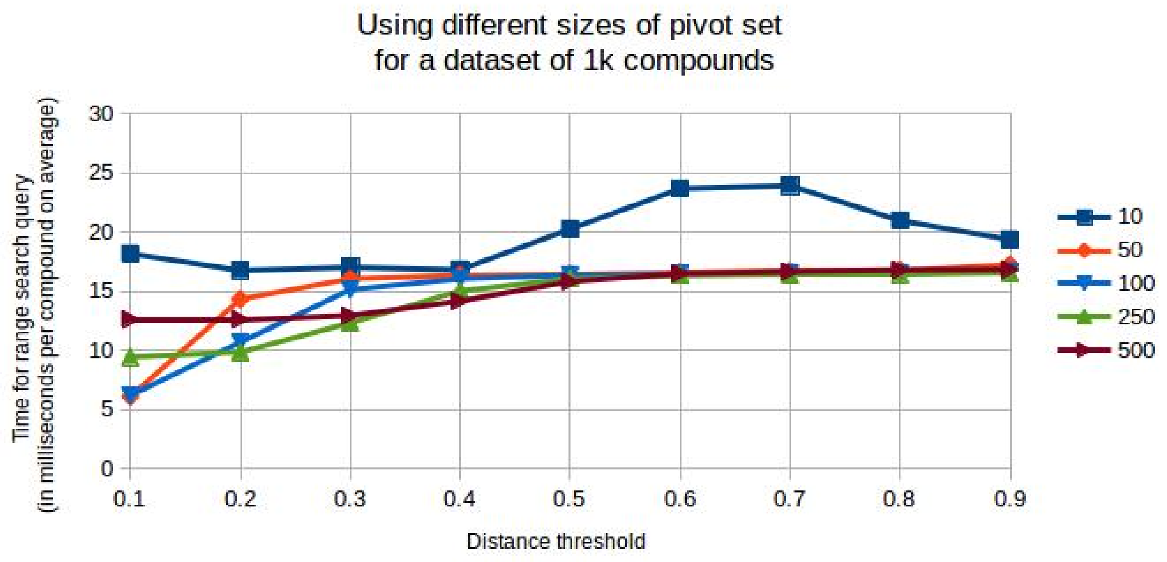

Figure 7.3: M-tree: Average Range Query time versus Distance Threshold for various sizes of pivot set - 1k database

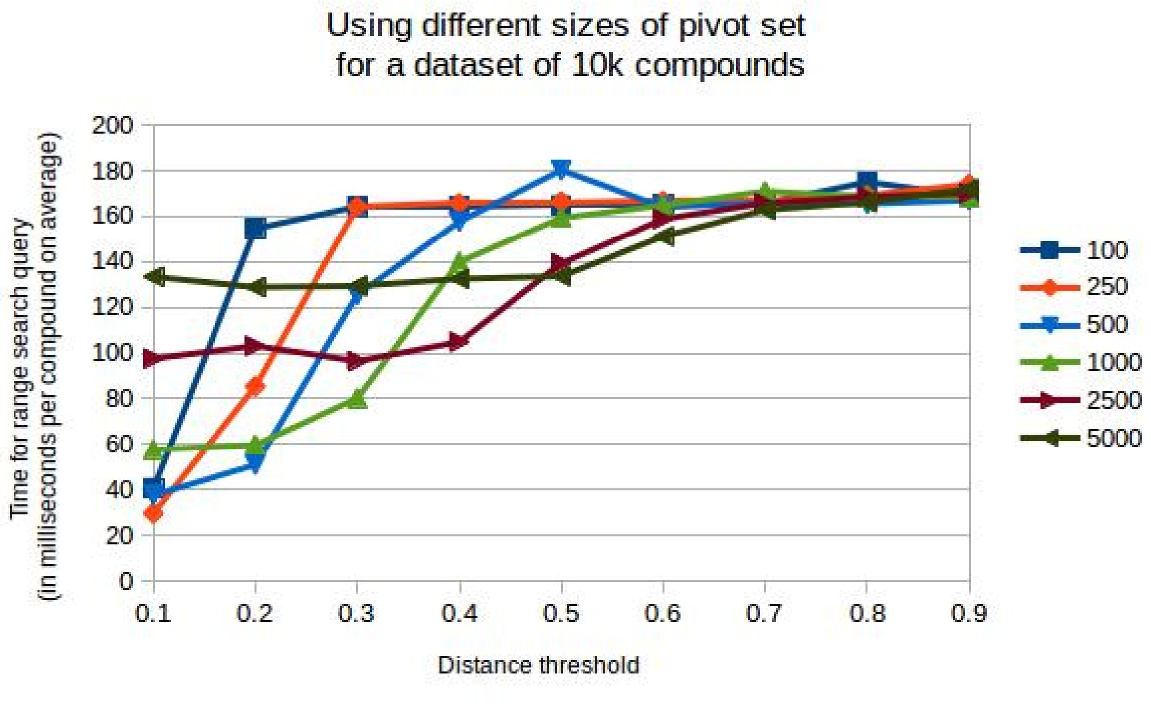

Figure 7.4: M-tree: Average Range Query time versus Distance Threshold for various sizes of pivot set - 10k database

There are two extremes here. If we use a very high number of pivots most of the time is spent doing computation involving triangle inequality bounds and hence it is not favourable. Similarly if we choose a very low number of pivots, it is not possible to exploit the bounds effectively to decisively prune or

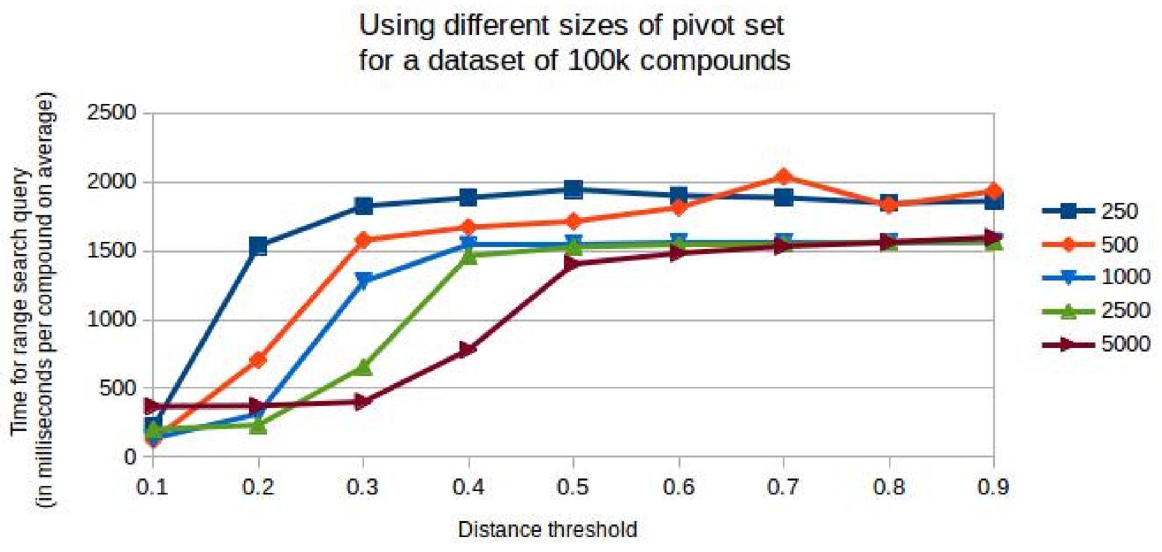

Figure 7.5: M-tree: Average Range Query time versus Distance Threshold for various sizes of pivot set - 100k database

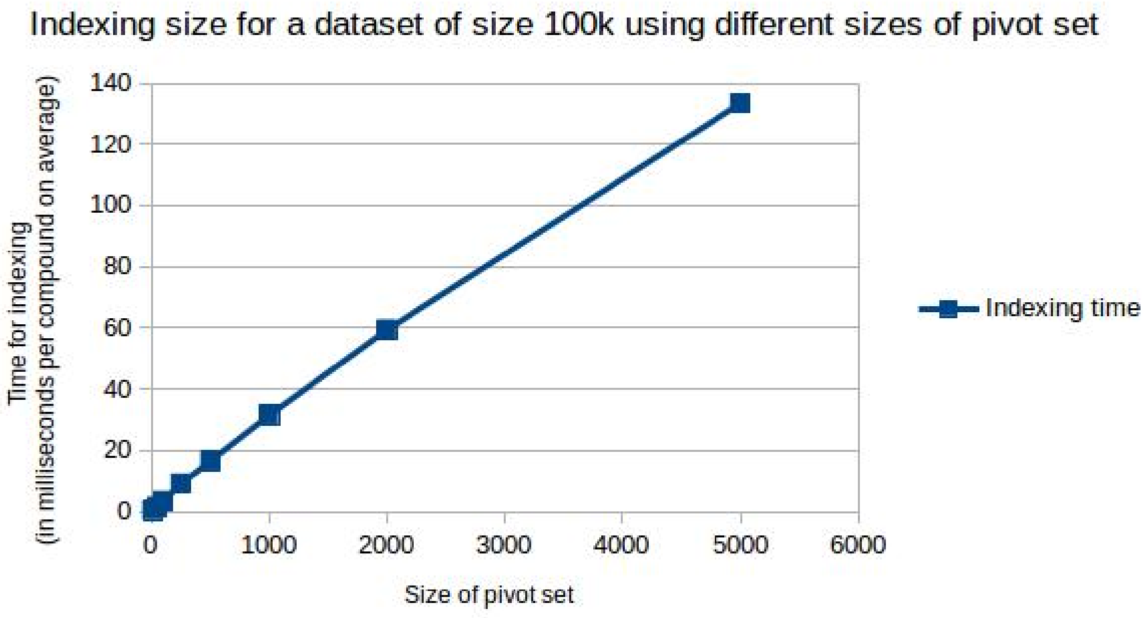

Figure 7.6: M-tree: Indexing time versus different sizes of pivot set size

include the subtrees from our answer result set. For low number of pivots used we notice that for high and low values of threshold the query time is lower than that for the middle range, where bounds doesn't help well enough.

As seen in 7.7, we observe that for low value of threshold at 0.1, the number of pivots to be chosen seems to have a minima between 500 and 1000 for a dataset size of 100k after which it increases . This is because there is a tradeoff between the time saved by pruning the subtrees versus the time utilized in checking if the nodes can be actually pruned. For a high threshold value of 0.9 it can be seen that we require more number of pivots .

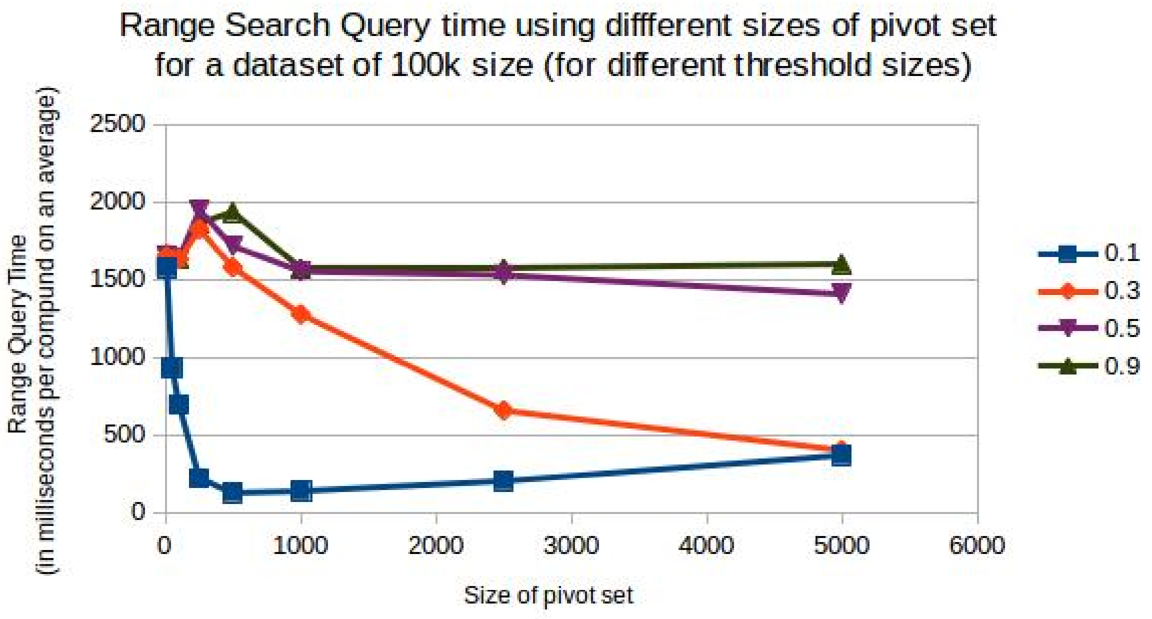

Figure 7.7: M-tree: Average Range Query time versus various sizes of pivot set for different theshold distances

7.1.4 Dataset Size

We varied the dataset size for the non-binary version of the Swamidass dataset to 1000, 10,000 and 100,000 and compared the indexing time as well as range search query time for various thresholds.

As seen in 7.9, we can observe that for a particular value of the distance threshold , the range search query time also increases linearly with dataset size . The slope is higher for higher values of threshold distance.

As seen in 7.8 we can observe that the average indexing time per compund on average increases linearly with increase in dataset size . This is as a result of our pivoting method with sampling which reduces the computation. If our pivoting step had been an exhaustive search on the database the curve would have not been linear,

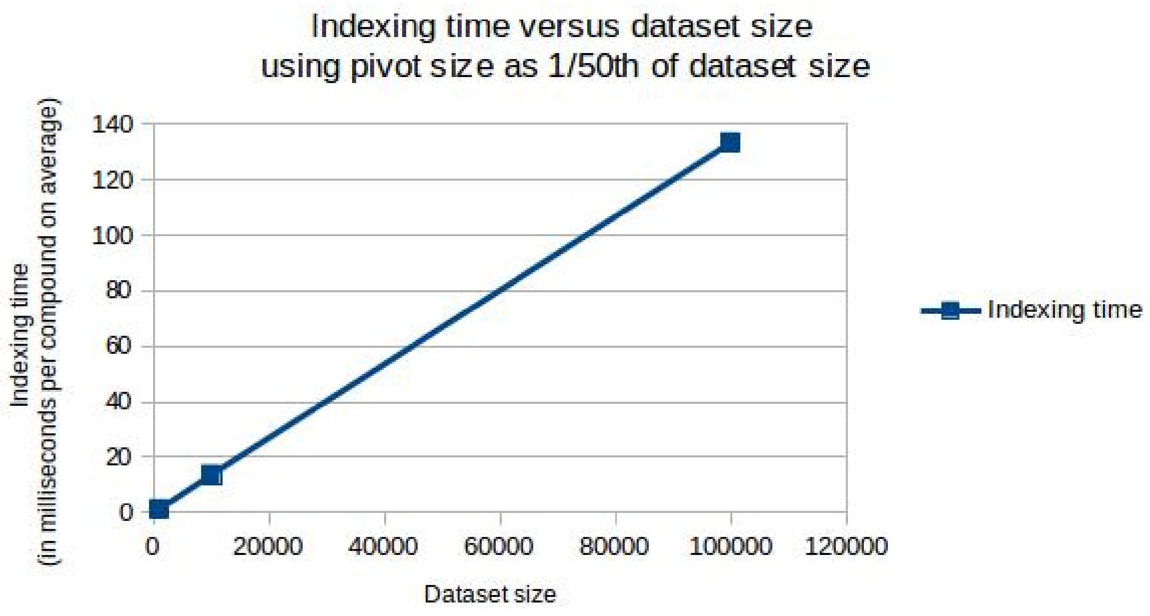

Figure 7.8: M-tree: Indexing time versus dataset size

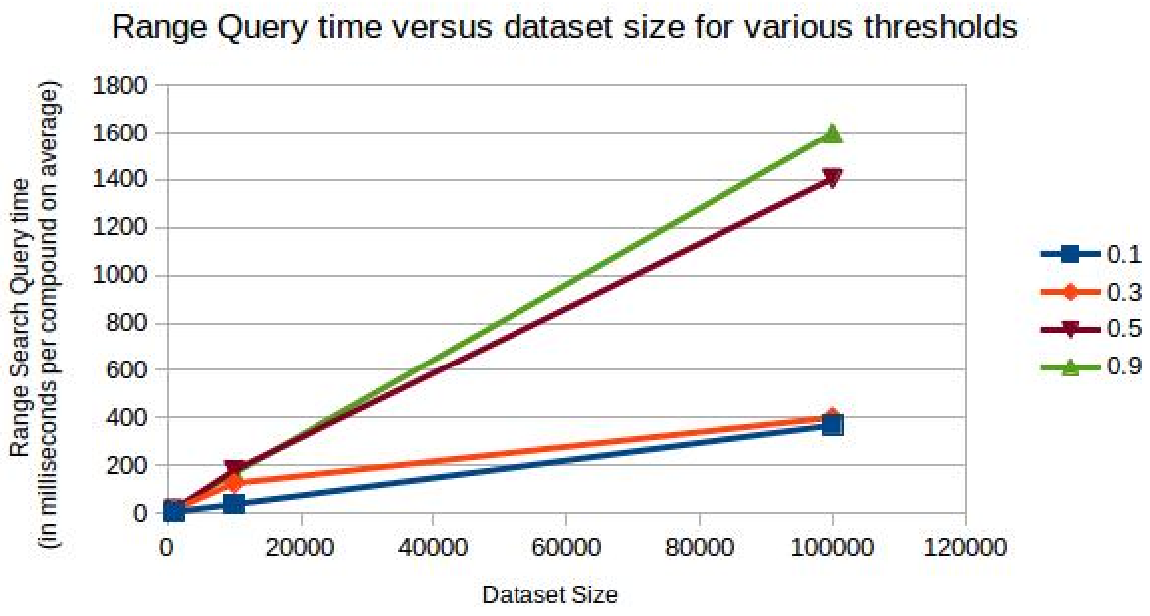

Figure 7.9: M-tree: Range Query time versus dataset size

7.2 Inverted Indexing Analysis

For the below experiments the testbed used was a server with 512GB RAM, 24TB disk space, and 2 quad core Xeon processor. We tested the inverted index structure against both binary as well as non-binary versions of the Swamidass dataset. For evaluation and verification processes, a complete linear database scan was done as with the M-tree case to compare the result sets. One advantage of Inverted indexing procedure over the M-tree index structure is the much lesser time required for the indexing step.

Indexing time per compound is observed to be almost constant between 1ms to 2ms per compound on average. As observed in 7.13 the graph is almost straight parallel to the x-axis for both the binary as well as the non-binary dataset.

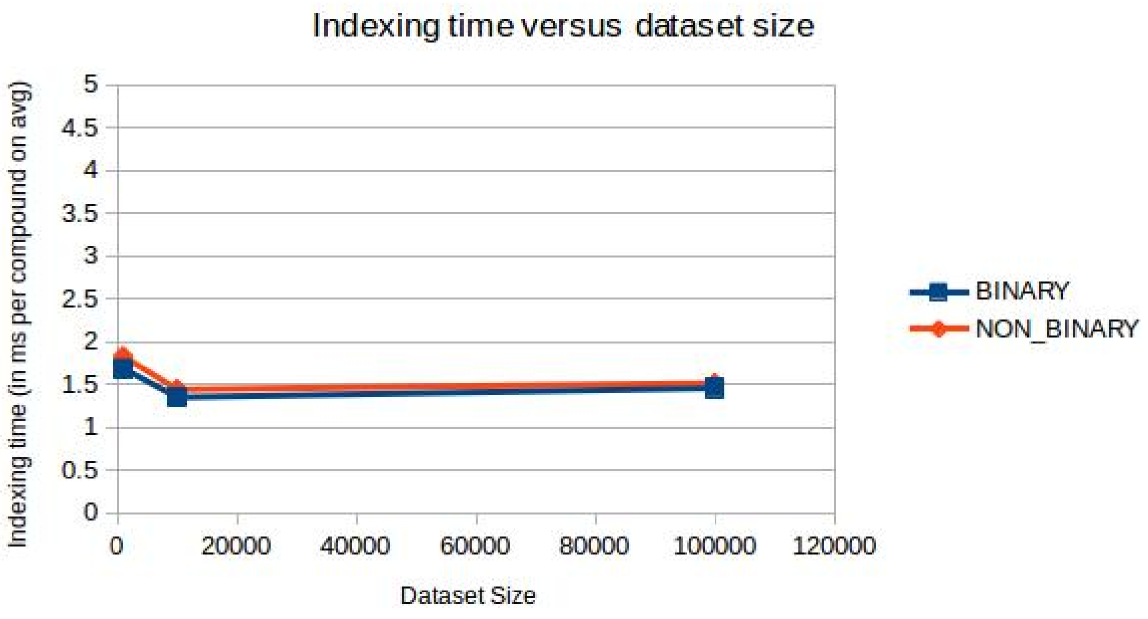

Figure 7.10: Inverted Index: Indexing time versus dataset size for Inverted Indexing

7.3 Comparison of our techniques

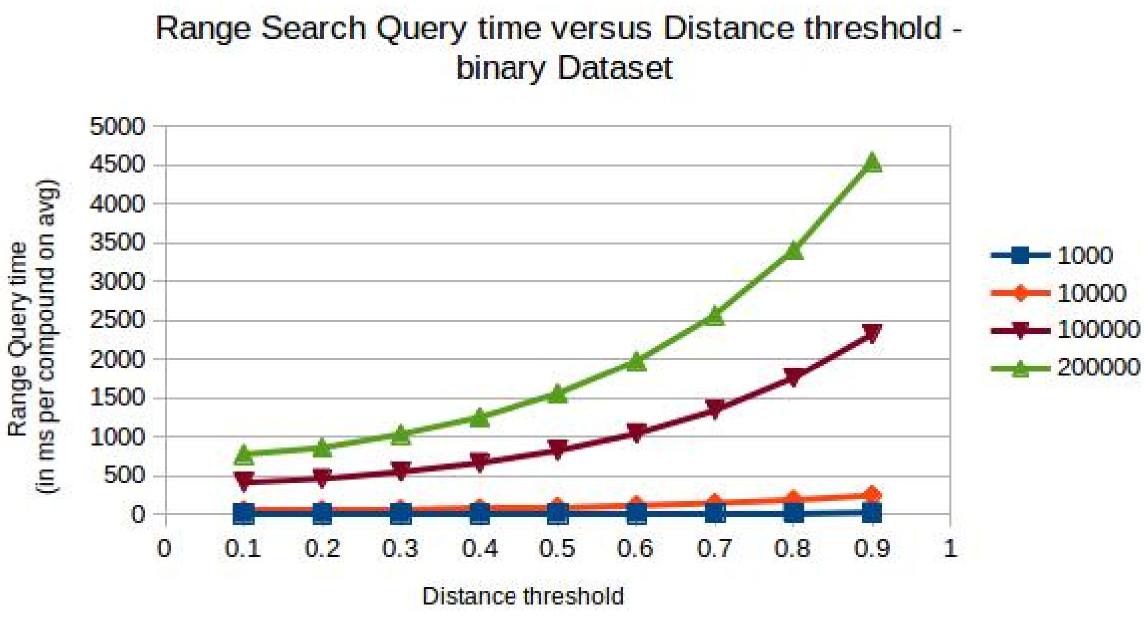

Figure 7.11: Inverted Index:

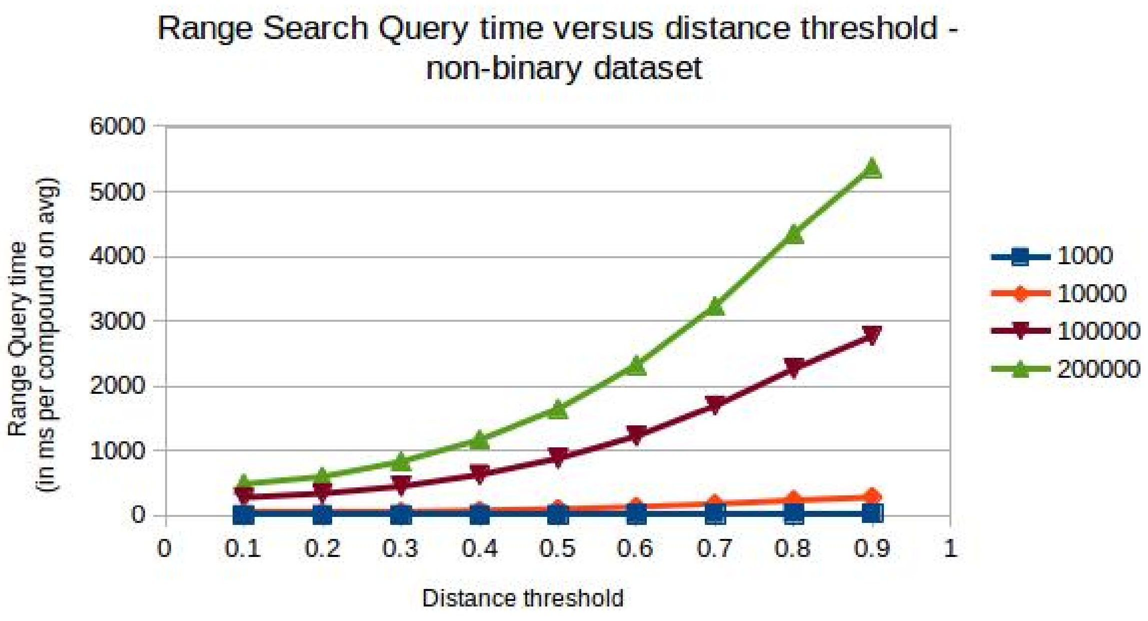

Figure 7.12: Inverted Index:

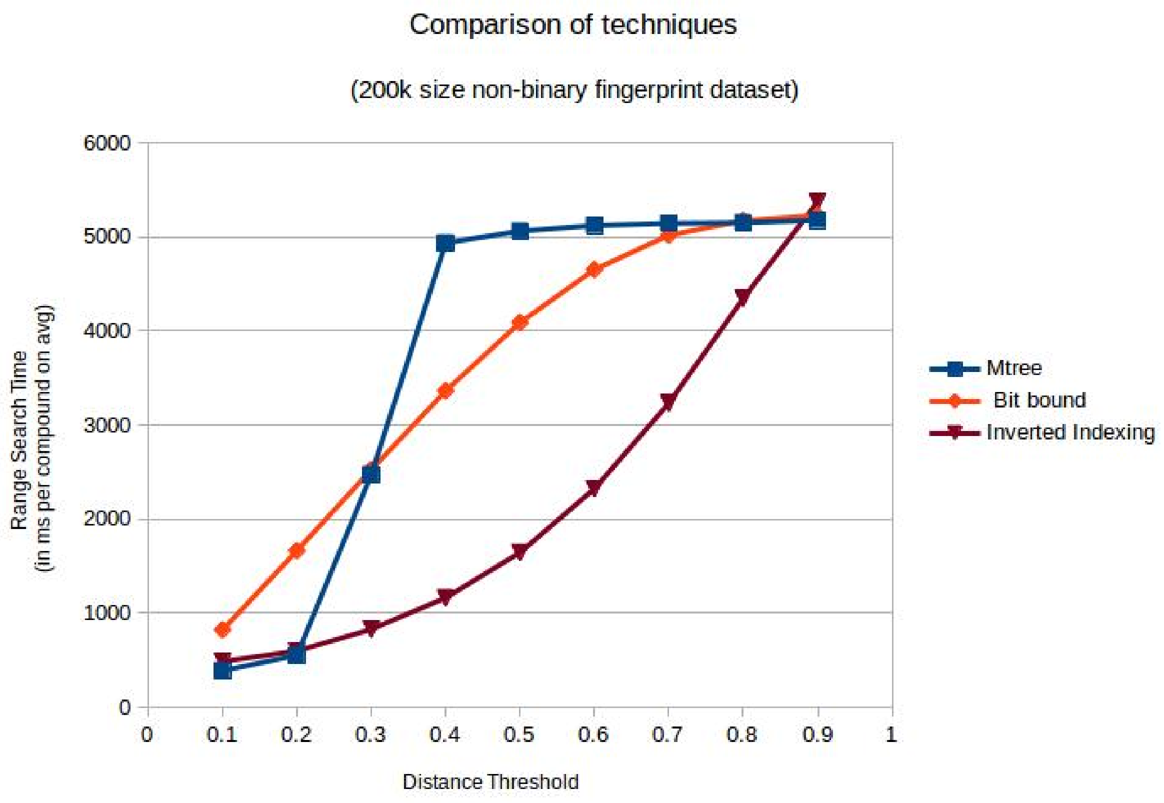

Figure 7.13: Comparsion:

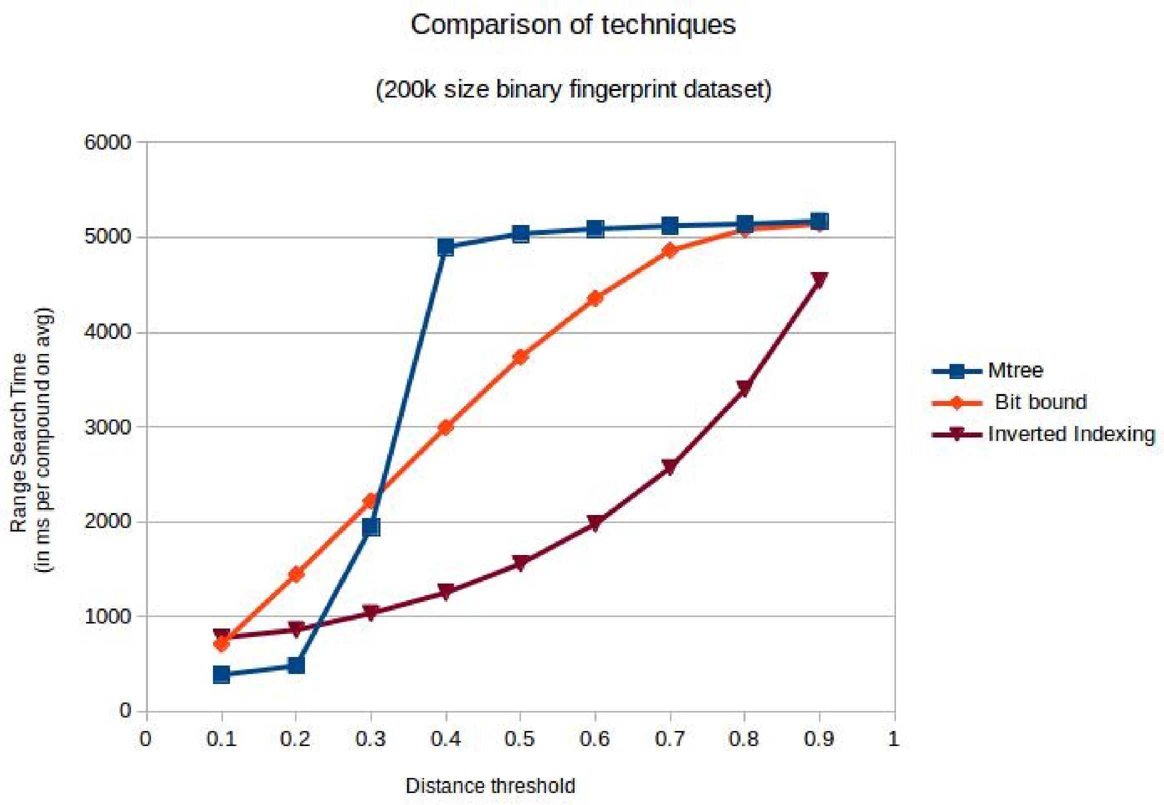

Figure 7.14: Comparsion:

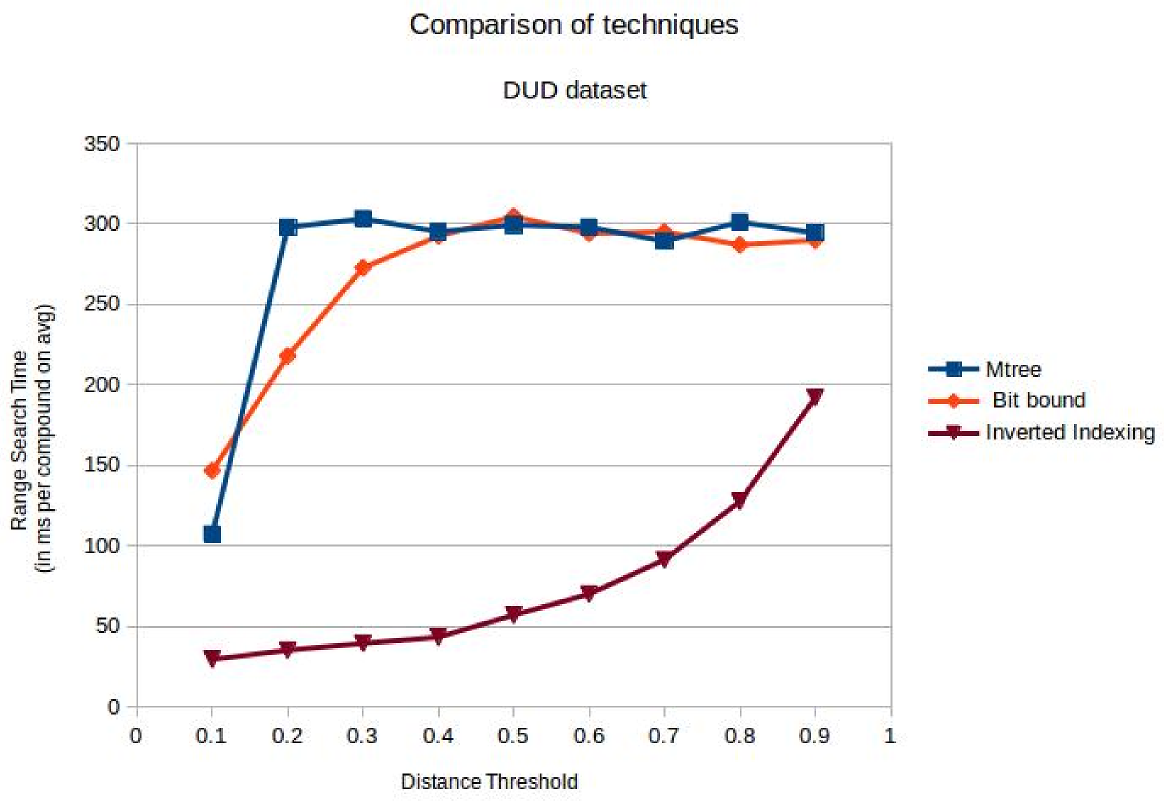

Figure 7.15: Comparsion:

REFERENCES

[1] **Alvarez, J.** and **B. Shoichet**, *Virtual screening in drug discovery*. CRC press, 2005.

[2] **Aung, Z.** and **S.-K. Ng**, An indexing scheme for fast and accurate chemical fingerprint database searching. *In Scientific and Statistical Database Management*. Springer, 2010.

[3] **Aung, Z.** and **K.-L. Tan** (2007). Rapid retrieval of protein structures from databases. *Drug discovery today*, **12**(17), 732–739.

[4] **Baldi, P.**, **R. W. Benz**, **D. S. Hirschberg**, and **S. J. Swamidass** (2007). Lossless compression of chemical fingerprints using integer entropy codes improves storage and retrieval. *Journal of chemical information and modeling*, **47**(6), 2098–2109.

[5] **Ciaccia, P.** and **M. Patella** (2002). Searching in metric spaces with user-defined and approximate distances. *ACM Transactions on Database Systems (TODS)*, **27**(4), 398–437.

[6] **Ciaccia, P.**, **M. Patella**, **F. Rabitti**, and **P. Zezula**, Indexing metric spaces with m-tree. *In SEBD*, volume 97. 1997.

[7] **Fligner, M. A.**, **J. S. Verducci**, and **P. E. Blower** (2002). A modification of the jaccard–tanimoto similarity index for diverse selection of chemical compounds using binary strings. *Technometrics*, **44**(2), 110–119.

[8] **James, C.**, **D. Weininger**, and **J. Delany** (2007). Daylight theory manual. 2004. *URL http://www. daylight. com/dayhtml/doc/theory/theory. toc. html.*

[9] **Lipkus, A. H.** (1999). A proof of the triangle inequality for the tanimoto distance. *Journal of Mathematical Chemistry*, **26**(1-3), 263–265.

[10] **Metwally, A.**, **D. Agrawal**, and **A. El Abbadi**, Efficient computation of frequent and top-k elements in data streams. *In Database Theory-ICDT 2005*. Springer, 2005, 398–412.

[11] **Nasr, R.**, **D. S. Hirschberg**, and **P. Baldi** (2010). Hashing algorithms and data structures for rapid searches of fingerprint vectors. *Journal of chemical information and modeling*, **50**(8), 1358–1368.

[12] **Swamidass, S. J.** and **P. Baldi** (2007). Bounds and algorithms for fast exact searches of chemical fingerprints in linear and sublinear time. *Journal of chemical information and modeling*, **47**(2), 302–317.

[13] **Willett, P.** (2006). Similarity-based virtual screening using 2d fingerprints. *Drug discovery today*, **11**(23), 1046–1053.