Indexing chemical fingerprints for efficient querying of molecular databases

A Project Report

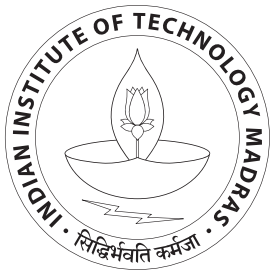
submitted by

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

I would like to express my sincere gratitude to Dr. Sayan Ranu, who has been a mentor and a friend. His support and guidance have been priceless. A special thanks to Jithin Vachery for sharing his knowledge and expertise. Many of the ideas described in this thesis would not have been possible without the many constructive and valuable discussions I had with the both of them.

I would like to thank all the remarkable professors in the Computer Science Department for their constant encouragement. A notable mention to all my friends for the wonderful memories over the course of the past five years here at the Indian Institute of Technology, Madras. In particular, Dheepikaa and Sahitya for being ideal project partners over the years, Venkat, Sudheer, Aahlaad, Dhanvin and Rajan for being the best hostel-mates, Arun and Chinmay, who were indirectly responsible for me shifting major to Computer Science and Dakshini, for proof-reading my thesis. Lastly, I would like to thank my very awesome mother, Soma, my always inspiring brother, Pratik and my twin sister, Abhilasha, who has been my guiding source of light.

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 data sets 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 (in terms of the answer set) when compared to a complete database linear scan .

The first technique uses an indexing technique based on the structure of the M-tree. 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 data set. 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-infomatics is largely concerned with the task of mining large chemical data-sets. 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. The direct comparison of chemical molecules (generally in the form of graph data) is difficult but conversion of the molecules into bit-strings or into vector format (known as the fingerprint of the chemical compound) makes it easier for researchers 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 as well as the time required to accomplish such a similarity detection task.

Chemical fingerprints are built based on compound properties like substructure features. In binary fingerprints each bit represents the presence or absence of a specific chemical property. For example the bits could represent the count of individual chemical atoms like carbon, oxygen, hydrogen or the number of saturated or unsaturated aromatic or non-aromatic rings. The reason why substructure presence is an important feature is because it can be used to reduce or filter out candidate mismatches in the sub-graph isomorphism test process. 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 include information like number of occurrences of particular substructures, presence of molecules, bonds between the molecules, etc. Generally the data-set is high dimensional and also highly sparse which inherently makes searching and indexing such chemical data sets challenging. To give an example, one of the data-sets we are working on is the PubChem data-set which contains about 264016 number of data points and 785985 total number of unique features. These numbers are representative of the high dimensional nature of the data-set. The sparsity of the data-set is indicated by the fact that the average number of features contained by a data point of the PubChem database is 270.602966.

For performing query operation on high dimensional data, it is generally observed that many conventional 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. The results of our range search techniques will be verified by comparing the range search result set against that obtained by a linear scan approach.

Similarity between chemical molecules plays an important role in various aspects of biology and chemistry like protein ligand docking, biological activity prediction, reaction site modeling and drug discovery. Furthermore, 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 process of calculating similarity between chemical compounds involves a number of steps. Firstly each chemical compound which is represented in a graph manner, with neighboring atoms, bond-linkages, substructures, etc have to be represented in a manner where finding similarity becomes simpler. When we represent items in a vector format, our job becomes easier with the use of measures like Jaccard Similarity as well as the Cosine Similarity. Hence the need to convert compounds to a format we described earlier known as fingerprints.

The conversion of the compounds to the vector format is known as fingerprinting. In the field of computer science, fingerprinting is a technique that hashes or maps every large data item to a much shorter string, generally a bit vector, also known as its fingerprint. This fingerprint is used to uniquely identify the data item for all purposes, very similar to how fingerprints of humans can be mapped uniquely for every single individual. Fingerprinting as mentioned earlier is used for avoiding transfer and comparison of large bulky data.

Various scoring schemes like Tversky, Pearson, Dice, and Kulczynski are available for comparing

two given fingerprints [17, 19]. 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

$$T_s(X,Y) = \frac{\sum_i X_i \wedge Y_i}{\sum_i X_i \vee Y_i} \tag{1.1}$$

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.

The first technique we used is that of M-tree [7] based indexing. In this technique we have proposed a novel pivot selection technique, which results in a efficient search time for range queries. We use triangle inequality bounds on the distance measure to effectively prune out compounds or alternatively include a whole set of compounds in the range search query answer set. We have proposed an indexing technique which runs in linear time against the size of the data-set. We have also proposed methods to take care of out-liers in the data-set during the indexing procedure .

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 inverted indexing idea comes from the use of the inverted index in information retrieval and text mining, where for every term, they list out the set of documents that the term occurs in. Similarly here, we record for every feature, the reference to the points which contains the feature (a non-zero feature value). We propose a unique pruning method which helps in cutting down the candidate list for our result set. The large size of the data and the sparsity of the data are the primary challenges which we have addressed in this work.

To give you an overview of the chapters; In chapter 2, previous work in this field of research have been summarized. We have also outlined the shortcomings of the state of the art techniques which we will address in our work. In chapter 3, we give the formal definition of our problem, study the utilization of the Tanimoto similarity measure and describe the exact threshold similarity search problem which we intend to tackle.

In chapter 4, a detailed description of the M-tree data structure has been given along with the steps proposed for our indexing procedure. We also explore improvisations on the baseline indexing technique and how Lipschitz embedding can be utilized in our indexing method. In chapter 5, the use of the inverted index has been analyzed, along with the derivation of the bounds required for the pruning of the features and few other extensions to the baseline indexing technique.

In chapter 6, we do a comprehensive analysis of the data-sets to help us understand and exploit the nature of the fingerprint data to our advantage. Finally in chapter 7, we discuss the experimental analysis, results and observations of the comparison of our techniques to the state of the art technique on various data-sets. In chapter 8, we present a short summary of our entire work and entail directions for future extensions to our 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 up 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 [17]. 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. This method is also known as the Bit-Bound technique.

A different approach to the same problem can be seen in Nasr, Hirschberg, and Baldi [15] 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 unfavorable bounds.

In Napolitano, Tagliaferri, and Baldi [14], the authors proposed an adaptive OR based criterion algorithm which is a technique based on adaptive reference points to fasten the process of querying on a large database of chemical compounds represented as binary fingerprints. They also proposed a unifying view between the context of reference points and the previously proposed hashing techniques like the Bit Bound technique. They evaluated their algorithm by simulating queries against an excerpt from the ChemDB. They showed how their adaptive method was 2-4 times better than the Bit bound technique.

In Klein, Kriege, and Mutzel [11], the authors proposed a unique indexing method based on a coupling of tree and cycle features to speed up sub-graph queries in graph databases. They created their index using an exhaustive feature enumeration method and used a fingerprint algorithm to store the characteristics of the compound graph. The CT-Index is a useful alternative to existing methodologies. They were able to show the scalability of their technique on large databases.

Nasr, Vernica, Li, and Baldi [16] propose DivideSkip, an inverted index algorithm for similarity search while in Thiel, Sach-Peltason, Ottmann, and Kohlbacher [18], the authors present an optimized inverted index algorithm for the calculation of all pairwise similarities on 2D fingerprints of a given data set. They propose an efficient and effective method for the calculation of Tanimoto similarity on binary fingerprints from a large database of chemical compounds by describing a memory efficient representation of fingerprints.

Many of the aforementioned papers present the evaluations of their algorithms only on binary fingerprints . While some of the methods don't extend to non-binary chemical fingerprints, some of the papers

just mention that their procedure works on non-binary fingerprints without showing any evaluation. We are looking at exact similarity searches of chemical compounds and not interested in the approximation of the result set. Hence we do not look into methods like Locally sensitive hashing which maximizes the probability of collision of similar items but does not guarantee it.

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 data-sets (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.

Problem 1. *Range queries on a chemical data-set can be defined as the process where, when given a fingerprint, say 'f', a similarity measure 'sim', a threshold distance $'\theta'$ and a database of chemical compounds D, we find the subset $S \subset D$ of all fingerprints, such that:*

$$S = \{g \mid g \in D, sim(f, g) < \theta\} \tag{3.1}$$

Our aim as mentioned earlier is to come up with a novel indexing scheme which would make the aforementioned task 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 [17] and many other state of the art techniques developed in this domain, we are working on non-binary vector fingerprints.

The indexing time i.e. the time required to index a compound on average is also a concern for us. For any new compound added to the database we would like to be able to add it easily to the index structure without undergoing any major time-consuming changes. We would like our algorithms to scale on

large data-sets so we will compare our average range search query times on different sizes of the data-set.

An important thing to note here, which is also described in detail in section 3.3 is that we are looking at exact threshold similarity searches and not interested in approximate or probabilistic techniques. In the next section we explain properties of the standard similarity measure used in cheminfomatics to compare chemical fingerprints, the Tanimoto similarity and its extension to non-binary data-sets as well as how it can be exploited well to our benefit.

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.

Definition 1. *In mathematical terms, for binary fingerprints X and Y, where X_i is the i^th bit of fingerprint X we define Tanimoto similarity as:*

$$T_s(X,Y) = \frac{\sum_i X_i \wedge Y_i}{\sum_i X_i \vee Y_i} \tag{3.2}$$

The Tanimoto coefficient measure for chemical data can also be defined as $\frac{x}{x+y+z}$, which is the number of features shared by the two compounds divided by the size of union of the non-zero features in the two compounds together . The variable x is the number of features (or number of 1-bits in a binary fingerprint) present in both the compounds while y and z represent the features present in one while not being present in the other. We can observe that the Tanimoto Similarity has a range from 0 to 1, where high coefficient scores indicates a greater similarity among the compounds than lower coefficient scores. One important thing to note here is that a score of 1 doesn't indicate that the compounds are the same, it just denotes the presence of similar substructures, bonds, molecules, etc in both the compounds i.e. their binary signature or fingerprint is identical.

For non-binary fingerprints we can extend the same Tanimoto coefficient by using non-zero values as 1 . Another alternative is to use the similarity measure known as Min-Max similarity.

Definition 2. *Min-Max similarity(M_s) between two fingerprints X, Y, where X_i corresponds to the i_{th} feature value of X, can be formulated as follows:*

$$M_s(X,Y) = \frac{\sum_i min(X_i, Y_i)}{\sum_i max(X_i, Y_i)} \tag{3.3}$$

If the fingerprints are binary, this similarity measure reduces to the Tanimoto coefficient or 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:

$$M_d(X,Y) = 1 - M_s(X,Y) = 1 - \frac{\sum_i min(X_i, Y_i)}{\sum_i max(X_i, Y_i)} \tag{3.4}$$

$$M_d(X,Y) = \frac{\sum_i |X_i - Y_i|}{\sum_i max(X_i, Y_i)} \tag{3.5}$$

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 [12]. The triangle inequality bounds helps us in the M-tree index structure where we can use the bounds to prune the entire sub-tree of fingerprints or alternatively include the whole tree in our data-set.

3.3 Exact Threshold Similarity Search

We are primarily interested in similarity searches which retrieve all compounds from a database of compounds which are similar to the query compound as defined by a fixed measure of distance or similarity. We are looking at exact threshold similarity searched for our range search query and would not like to approximate our results. Non exact threshold similarity searches can be thought of as those algorithms which allow room for small false positive rates (where a false candidate arises in the result set). But

we would want to avoid this . One important reason for this is that these algorithms are essentially used to find drugs and compounds which contain desired properties. When a candidate drug is discovered, millions of dollars are spent in laboratory for experiments and trials before it can see the light of day. Drugs can be rejected because of high molecular weight, unstable bond structures, high reactivity, etc and hence most work done in this field of cheminfomatics has been on exact threshold similarity searches.

The M-tree index structure proposed in this work has a lot of parallels to hierarchical clustering with outlier detection. Conventional clustering methods fail due to the high dimensional nature as well as sparseness of a typical fingerprint data-set. The high dimensionality of the data-set is a primary concern of ours since it limits the scalability of our techniques. Applying Principal Component analysis does not guarantee us zero false positives.

We would like to produce a mapping onto a lower dimensional space where we can achieve a contractive mapping. A contractive mapping is required to guarantee no false positives. A contractive mapping is achieved when distance between two compounds described by the new distance metric in the lower dimensional space is less than or equal to the actual distance as described by the original distance metric in the higher dimensional original space. It can be formally defined as follows.

Definition 3. *Given a D-dimensional space X, and a lower L-dimensional space Y and distance metrics d_x and d_y defined respectively in the two spaces, a contractive mapping m from X $\rightarrow$ Y (mapping a compound from D -dimensions to L-dimensions) is such that for every pair of compounds a and b :*

$$d_x(a, b) > d_y(m(a), m(b)) \ \ \forall a, b \in X \tag{3.6}$$

If a contractive mapping can be achieved, we can use our indexing technique on the lower dimensional space so that the curse of dimensionality can be broken. For range queries, we can obtain a candidate set from the lower space and verify it in the higher dimensional space. The idea is that if the compound is actually within the threshold in the higher dimension, it will also be within the threshold with the new distance metric in the lower dimension owing to the contractive nature of the mapping. An example of a contractive mapping is Lipschitz embedding.

An inverted index structure can be exploited because standard chemical data-sets are known to have the feature presence (features against the number of points they appear in) follow a sort of power law distribution. And we are guaranteed to have only the common features contributing to similarity, hence the number of features to be considered is not very high. This is because on average every point has very few features compared to the total feature set size.

CHAPTER 4

M-tree based Index Structure

In this chapter, we explain the M-tree data structure along with an unique indexing process which builds on the base structure of the metric tree. We explore an alternate indexing technique as well. Later we describe the range search query procedure in detail and study a dimensionality reduction technique to help overcome the high dimensionality and sparsity of the data-sets.

4.1 M-tree

M-tree, also known as the Metric tree is a tree data structure constructed using a metric distance measure, and which relies on the triangle inequality for efficient range search queries. Similar to other tree data structures, an M-tree data structure has Leaf Nodes and non-Leaf nodes. Every non-leaf node has a pointer to its parent node, a pointer to its sub-tree, its object information and a covering radius denoting maximum distance of a node to any node in its sub-tree. 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 sub-tree 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 sub-tree and rooted at router object O_r is at at-most $r(O_r)$ distance to it i.e. $d(O_j, O_r)r(O_r)$. M-tree organizes the metric space into a set of, possibly overlapping, regions, to which the same principle is recursively applied.

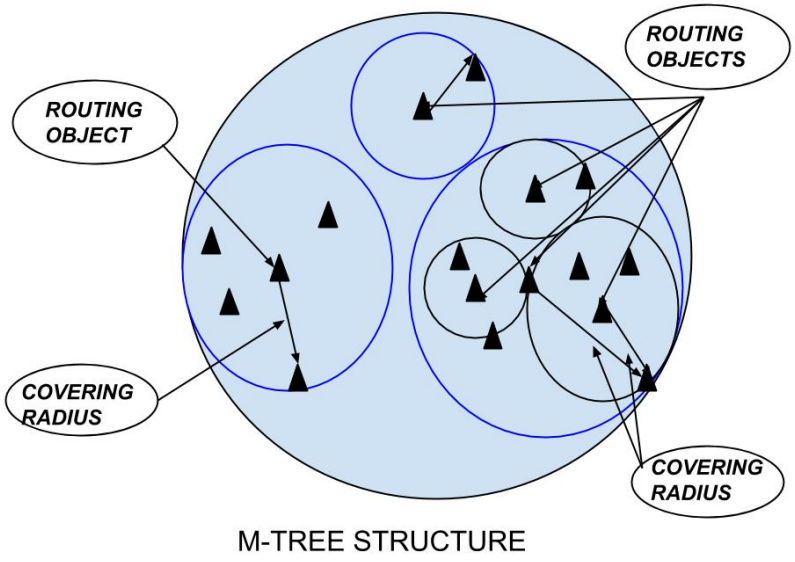


Figure 4.1: M-tree: Structure overview

4.2 Information in our data structure

We use a modified data structure of the M-tree. We record 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 sub-tree 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. To summarize, the following information is stored in each entry of a node in our tree:

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

4.3 Baseline Indexing approach

In the baseline indexing approach, we are partitioning the data-set 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.

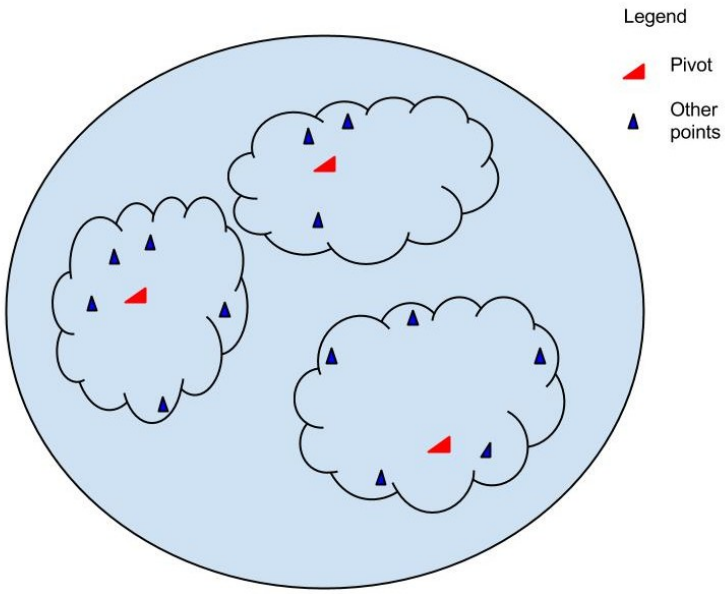


Figure 4.2: M-tree: Assigning points to a cluster after the pivoting step of the M-tree indexing process

1. Select the given number of random pivots (lets say p) from the database of chemical compounds. The number of random pivots selected in 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 to it i.e. the pivot with which it shares the highest Min-Max similarity. This is described in Figure 4.2.
3. The next step involves ranking all chemical compounds (which were assigned to a corresponding pivot) in order of their similarity to their closest pivot.
4. We compute the median similarity (each one to its closest pivot) 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 are marked as outliers in this step.

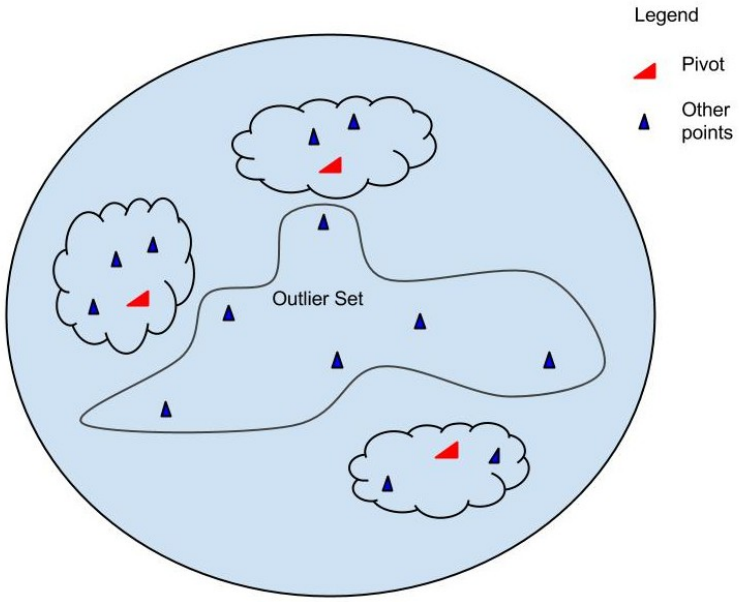


Figure 4.3: M-tree: Finding the outlier set in the M-tree index structure

6. The outliers are unassigned from the respective pivot sets and assigned to the ”outlier” pivot set. This has been described in Figure 4.3.
7. We recursively apply this technique (from step 1 to step 6) on the outlier set until the outlier set size reaches a base limit (o), which is also a parameter to our algorithm. Once the size of the outlier set falls below o, we terminate this indexing process.

4.4 Alternate indexing approach

We tried an alternate approach where instead of choosing outliers we recursively apply the technique of assigning a closest pivot on each set. In simpler words, we cluster at each step and try to create numerous clusters among each cluster recursively. 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. Select the given number of random pivots (p) from the set. The number of random pivots chosen at this step is again a parameter to the experiment as previously mentioned.
2. The second step is similar to the earlier indexing technique. After selecting pivots we assign every other chemical compound in the database to one of the pivots based on the similarity to the pivots, just as before.
3. Instead of choosing outliers now, we work with the same sets and recursively apply this technique (steps 1 and 2) on the respective sets until the corresponding set sizes reaches a base limit .

4. The set size limit (o) is fixed as the number of pivots chosen at each step (p) i.e. the method is applied recursively on the corresponding sets until the size of the set is lower than the pivot set size disabling us from selecting pivot sets.

Through experiments conducted on the PubChem data-set (which will be described later), we concluded that our previous indexing technique was more effective than this indexing method. Hence we will be using the earlier technique for all experimental evaluations. A reason for this indexing technique failing to provide us with better results, is that the tree was becoming imbalanced, causing the height of the tree to increase as well as the querying time.

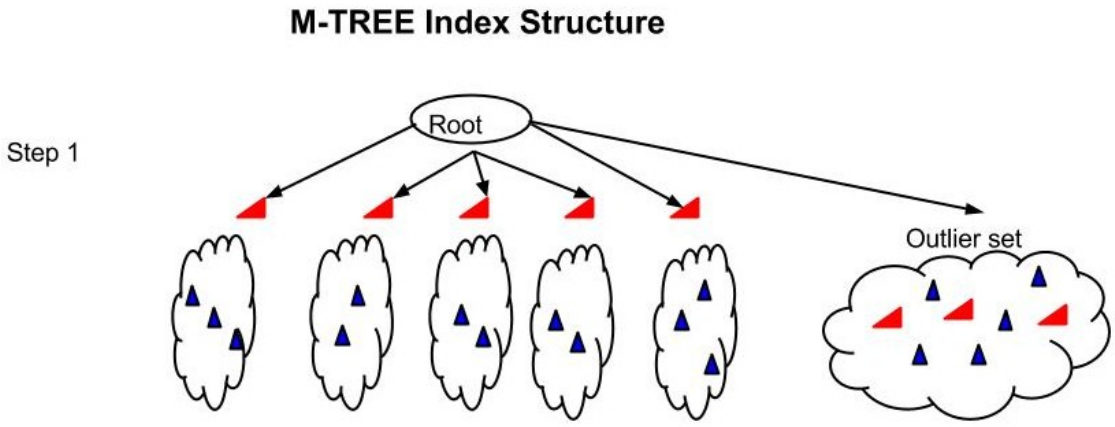


Figure 4.4: M-tree: Tree structure Step 1

4.5 Selecting pivots in the indexing process

Instead of randomly selecting 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 idea is that the clusters (with the pivots as cluster centre), hence formed, would be spread out. The procedure can be explained as below:

1. Select a random point.
2. The second point is selected in such a manner, that it is farthest from the point selected first. This will require a full database scan.

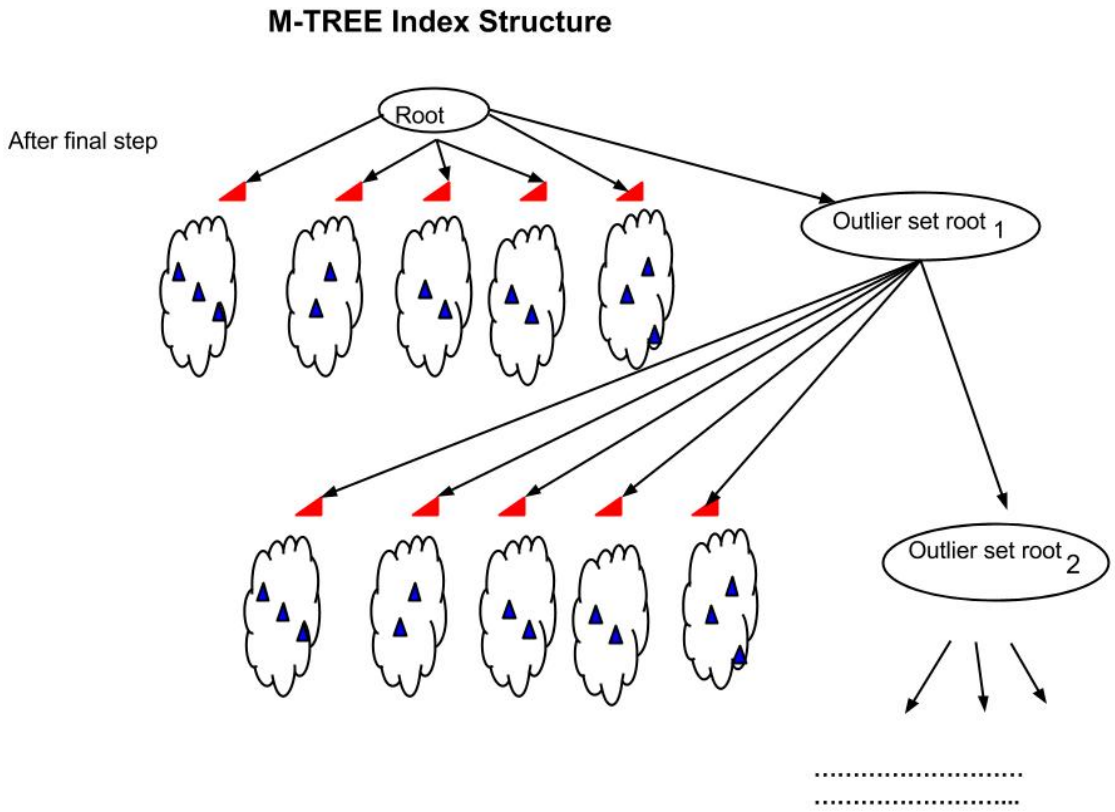


Figure 4.5: M-tree: Tree structure - After final step

3. The third point is chosen such that its minimum distance to either of the previous two pivots is maximized.
4. This process continues. At the i^{th} iteration, the i^{th} pivot 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 would need to compute an all pair similarity initially. At the i^{th} step, the remaining $n-i+1$ points in the database need to compute its minimum distance to the $i-1$ points already present in pivot step. Finally select the point among the $n-i+1$ points with the maximum computed distance. The computation complexity of this pivot selection method is of the order as shown below (Here n is the size of the entire database and p is the number of pivots to be chosen at each step of the indexing process):

$$T(n) = n^2 + \sum_{i=1}^{p}((n-i+1)(i-1) + n - i + 1) \tag{4.1}$$

$$T(n) = O(n^2 + np^2) \tag{4.2}$$

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 (chosen empirically), with approximately about 20,000 points in the subset. Since we have a fixed set of points (let this number be k), our computation time complexity for selecting the pivots decreases as shown below:

$$T(n) = k^2 + \sum_{i=1}^{p}((k-i+1)(i-1) + k - i + 1) \tag{4.3}$$

$$T(n) = O(k^2 + kp^2) \tag{4.4}$$

4.6 Range Search Querying

Given a query chemical fingerprint q and a distance threshold θ we want to find the set of chemical fingerprints which satisfy the conditions of the range search query. We exploit the triangle inequality for the same. The basic idea is to be able to prune sub-trees based on the covering radius of the pivot of the sub-tree and the distance of the query to the pivot. The procedure for the range search querying can be described by the following steps:

1. Let the query fingerprint be q and the fingerprint pivot be p_i of sub-tree 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).
2. 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 $max(dist(q, p_i)$ - $r_i, 0)$.
3. Hence we can compute the range of the distance of any node in S_i to q.
4. If the upper bound of the range or the maximum distance is lesser than the threshold distance θ, we can add all the nodes of the sub-tree S_i to our resultant set.
5. If the lower bound of the range is greater than the threshold distance θ, we can prune the sub-tree S_i, since we can say with certainty that the distance of every node in the sub-tree S_i to the query point is greater than the threshold θ.
6. If there is an intersection in the intervals we recursively apply this technique on the second level of children in the sub-tree S_i until we reach a leaf node. We make each of the children of the root of the sub-tree S_i as the new p_i and repeat the steps in each of the corresponding sub-trees.

4.7 Lipschitz Embedding

The performance of the M-tree is limited by the fact that the data is very sparse. This results in loose clusters being formed during the indexing phase. To improve upon this we performed a dimensionality reduction of the data into another space using Lipschitz embedding. Lipschitz embedding results in a contractive mapping. Let us define the basic terms before we can explain how to fit the embedding process into our indexing procedure.

Definition 4. *Embedding is the transformation from one space and distance function to another space and distance function. (The new space is generally a vector and the new distance function is, more often than not, Euclidean or some other L_p norm distance function).*

Definition 5. *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, as described in detail below.*

The Lipschitz embedding procedure can be formally explained 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 defined as:

$$d(p_j, A_i) = min_{a \epsilon A_i} d(p_j, a) \tag{4.5}$$

4. The new feature vector f(p_j) i.e. the mapping of the object p_j in the new space, is then defined as:

$$f(p_j) = \frac{d(p_j, A_1)}{k}, \frac{d(p_j, A_2)}{k} ... \frac{d(p_j, A_k)}{k} \tag{4.6}$$

5. We define the new distance $d'(f(p_j), f(p_k))$ as :

$$d'(f(p_j), f(p_k)) = \frac{1}{k} \sum_{i=1}^{k} |d(p_j, A_i) - d(p_k, A_i)| \tag{4.7}$$

Note that the new distance measure is the L_1 norm in the new space.

$$L_1(f(p_x), f(p_y)) \ = \sum_{i=1}^{k} |\frac{d(p_x, A_i)}{k} - \frac{d(p_y, A_i)}{k}| \ = \frac{1}{k} \sum_{i=1}^{k} |d(p_x, A_i) - d(p_y, A_i)| \ = d'(f(p_x), f(p_y)) \tag{4.8}$$

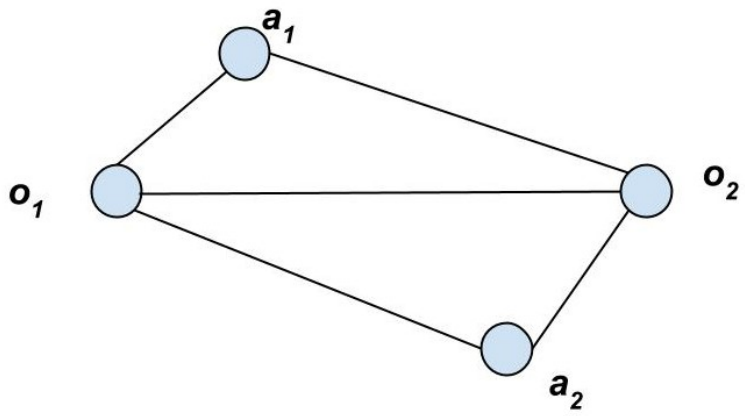


Figure 4.6: Lipschitz: Contractive mapping reference diagram

Theorem 1. *A Lipschitz embedding produces a contractive mapping i.e. a Lipschitz embedding of the form* $o_i \Rightarrow f(o_i)$ *with* $f(o_i)$ *as defined in Equation 4.6 and a new distance measure* d' *as defined in Equation 4.7 ensures the following:*

$$d'(f(o_i), f(o_j)) < d(o_i, o_j) \; \forall \; i, j \tag{4.9}$$

Proof. Consider any arbitrary reference set A_l.

$$\exists a_1 \in A_l, d(o_1, A_l) = d(o_1, a_1) \tag{4.10}$$

$$\exists a_2 \in A_l, d(o_2, A_l) = d(o_2, a_2) \tag{4.11}$$

as seen in Figure 4.6.

Now consider $|d(o_1, A_l) - d(o_2, A_l)|$

$$
\begin{aligned}
|d(o_1, A_l) - d(o_2, A_l)| &= |d(o_1, a_1) - d(o_2, a_2)| \\
&= max(d(o_1, a_1) - d(o_2, a_2), d(o_2, a_2) - d(o_1, a_1)) \\
&\leq max(d(o_1, a_2) - d(o_2, a_2), d(o_2, a_1) - d(o_1, a_1)) \\
&\leq d(o_1, o_2)
\end{aligned} \tag{4.12}
$$

Now consider $d'(f(o_i), f(o_j))$ and using Equation 4.7 and Equation 4.12

$$
\begin{aligned}
d'(f(o_i), f(o_j)) &= \frac{1}{k} \sum_{l=1}^{k} |d(o_i, A_l) - d(o_j, A_l)| \\
&\leq \frac{1}{k} \sum_{l=1}^{k} d(o_i, o_j) \\
&\leq \frac{1}{k} (k d(o_i, o_j)) \\
&\leq d(o_i, o_j)
\end{aligned} \tag{4.13}
$$

Hence proved that Lipschitz embedding is a contractive mapping. □

Theorem 2. *A Lipschitz embedding of the form $o_i \Rightarrow f(o_i)$ with $f(o_i)$ as defined in Equation 4.6 and a new distance measure d' as defined in Equation 4.7 ensures that d' is a metric in the new space.*

Proof. We have already shown from Equation 4.8 that d' is equivalent to the L_1 norm in the new space. Since we know that L_1 norm is a metric, it follows the new distance measure d' is also metric in the new space. □

4.8 Using Lipschitz Embedding for Range Search

We have chosen singleton sets as the reference sets in our embedding process for simplicity and to reduce computation. The choice of these points is done such that they are maximally far apart, i.e., the 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 idea that farthest

points can be used as a more ideal representative of the data-set, rather than points close together. Since choice of such an optimal set is NP-hard, we perform a randomized set construction.

1. Select 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 (based on the criteria explained above, of having the largest minimum distance to the already selected reference sets).
4. Add the point to the reference set.
5. Repeat step 3-4 until Reference set is full

Using Lipschitz embedding we propose the following indexing and range search process:

1. Embed the data-set using Lipschitz embedding and the pivoting method described above.
2. Follow the M-tree based indexing process but in this new embedded space.
3. Carry out the range search to compute the candidate set. Since from Theorem 1, we know that Lipschitz is contractive we are assured to have our candidate set as a super set of the actual range search result .
4. We do a similarity computation of all the points in the candidate set to the query, in the original space, to verify if the candidate point is not a false positive.

Through experiments performed on the PubChem dataset (which will be described later), we observed that our performance had deteriorated from that of the baseline technique. This is because the candidate set, which we compute in step 3 is very large, causing us to do a lot of unnecessary similarity computation.

CHAPTER 5

Inverted Index Structure

The high dimension and sparsity of the typical chemical fingerprint data-set are an impediment to our indexing and range search process. In this chapter, we will describe our construction of an inverted index structure on features. The inverted indexing idea comes from the use of the inverted index in information retrieval and text mining, where for every term, they list out the set of documents that the term occurs in. Similarly here, we record for every feature, the reference to the points which contains the feature (a non-zero feature value). Also, chemical data (or fingerprint data) is known to have an almost power law distribution for the presence of features in the data points and we wish to exploit this. We explain how the inverted index is an obvious choice for point queries, propose a novel indexing structure using bounds on the maximum similarity and also provide an extension to our algorithm to non-binary fingerprints.

5.1 Indexing Process

We have an inverted index on the features. For every feature we keep a hash table which stores a reference to every data containing the particular feature. Depending on the type (binary or non-binary) of the fingerprint, the indexing process might require pre-processing of the following values; for every feature f_i belonging to the set of features seen in the database, a storage of the minimum and maximum number of features present in any point containing the particular feature f_i, minimum non-zero and maximum value taken by any point containing the feature f_i, as well as the minimum sum of feature values among all points containing the particular feature f_i. All the aforementioned data has been stored in hash tables.

5.2 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 data-set 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 .

5.3 Range Query and K-nn

Range query and K-nearest neighbour searches are much more complicated queries, when compared to a point query. The candidate set in this case is much more diverse and distributed. For a range search query, every point, which has atleast one common feature with the query point is a potential candidate for our result set. The candidate set for a Range query will be the union of all the point lists associated with the features of p, which in the worst case is of the order of N. K-nn search (or the top-k most similar compounds) is not explored in this work. An idea which can be explored and has been discussed briefly in earlier work is as follows: We use Locally sensitive hashing methods or other approximation techniques to come up with a set points which collide with the query (or are similar to the query). This set of points will help us come up with an estimate on a threshold similarity, which can then be used for a range search for obtaining a candidate set.

5.4 Baseline technique for Range Search

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 scan on the compounds present S_q and compute its similarity to the query.
5. The range search result set is a subset of S_q, containing points which fall within threshold t distance from the query.

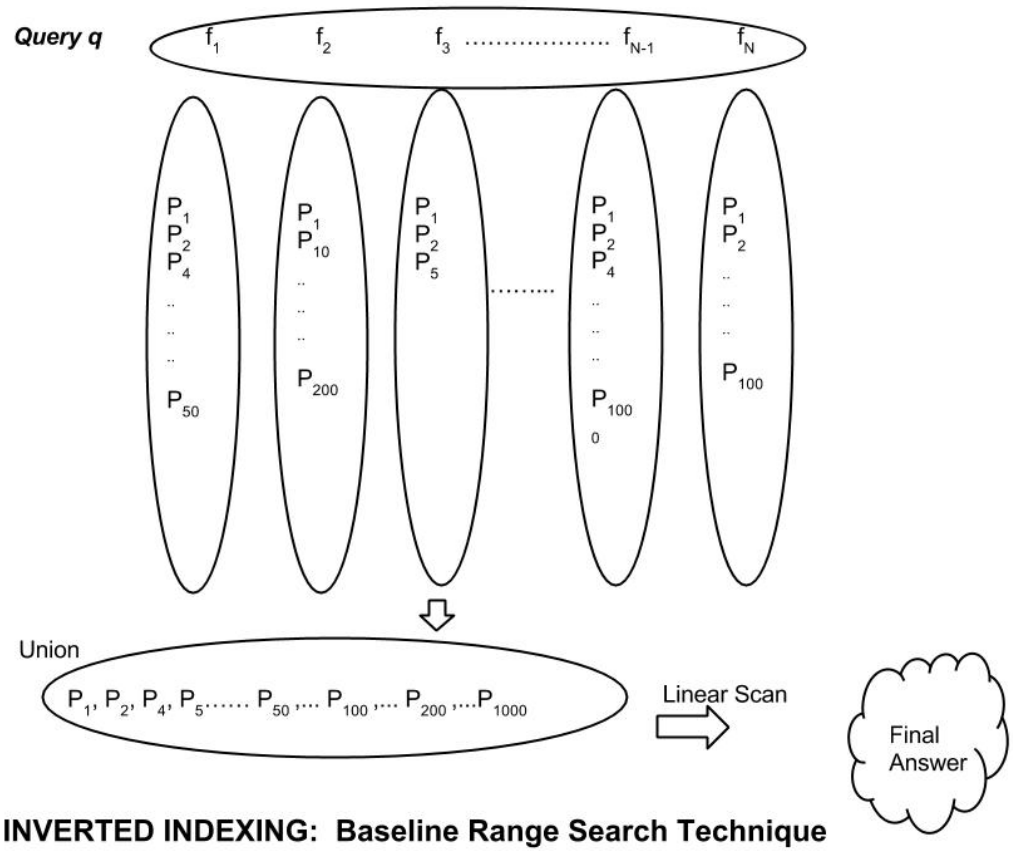


Figure 5.1: Inverted Indexing: Procedure Diagram

5.5 A Pruning bound for binary fingerprints

We state theorems in this section which will derive the bounds we utilize in our proposed range search technique described in the section 5.6. The technique will build on the baseline algorithm mentioned in section 5.4.

Pruning of a feature can be described as ignoring all the points, containing the particular feature, from the candidate result set of the range search query process. The point is pruned out for that particular feature only. If it contains another feature, which is common with the feature set of the query, it can be present in the candidate set. Pruning out a feature, which contains many number of points, will help save on the computation time spent on similarity calculation of the database compounds with the query.

Theorem 3. *For a case of binary fingerprints, given a query q and a threshold t, consider the feature f_i. If P is the set of points from the database which has the feature f_i set to 1, then we can prune all such points p_j of P from being present in the candidate range search set for query compound q if the point p_j has only f_i in common with the query q and it follows the following bound.*

$$\frac{1}{N_q - 1 + V_i} < 1 - t \tag{5.1}$$

where N_q is the number of features in query q and V_i is the minimum number of features present among all points having the feature f_i

Proof. Our assumption is that a point p_j from P is such that it has only f_i in common with the query q

The maximum Tanimoto similarity $S_{max}(p_j, q)$ such a point p_j can have with the query point q is given by

$$S_{max}(p_j, q) = \frac{1}{N_q - 1 + V_i} \tag{5.2}$$

This is because the Tanimoto similarity as described in Equation 3.2 is such that the numerator equals the number of common features which according to the assumption is 1. The denominator in the above equation is the minimum size of the union of the feature set of any such point p_j from P and the query point q. The equation gives us the right hand side of the bound in Equation 5.1.

The minimum similarity S_{min} required with the query for a candidate compound to be in the result of the range search is 1-t where t is the maximum distance threshold.

If the following equation were to hold:

$$S_{max}(p_j, q) < S_{min} \tag{5.3}$$

Then maximum possible similarity of a point would be less than the minimum required similarity for the point to be considered a candidate, hence we could prune the set of all such points.

From Equation 5.1, Equation 5.2 and Equation 5.3, we get the following bound,

$$\frac{1}{N_q - 1 + V_i} < 1 - t \tag{5.4}$$

which is as the theorem states. □

Now let us consider pruning of multiple features together.

Theorem 4. *For a case of binary fingerprints, given a query q and a threshold t, consider the feature set $F = f_1, f_2, ...f_M$. If P is the set of points from the database which has atleast one of the features f_k*

($f_k \in F$), set to 1 then we can prune all such points p_j of P from being present in the candidate range search set for query compound q if the point p_j does not have any other common feature other than in the set F, and it follows the following bound.

$$\frac{M}{N_q - 1 + \min\limits_{i \in (1,M)}(V_i)} < 1 - t \tag{5.5}$$

where M is the number of features in the set F, N_q is the number of features in query q and V_i is the minimum number of features present among all points having the feature f_i.

Proof. The proof is similar to how Theorem 3 is derived.

Our assumption is that a point p_j from P is such that it can have only features from the set $F = f_1, f_2, ...f_M$ in common with the query q

The maximum Tanimoto similarity $S_{max}(p_j, q)$ such a point p_j can have with the query point q is given by

$$S_{max}(p_j, q) = \frac{M}{N_q - 1 + \min\limits_{i \in (1,M)}(V_i)} < 1 - t \tag{5.6}$$

According to our assumption, the maximum possible value for the numerator in the score for Tanimoto similarity as described in Equation 3.2 is M when all features from the set are common to the query and the point p_j. The denominator in the above equation gives the minimum size of the union of the feature set of any such point p_j from P and the query point q. The rest of the proof is similar to how it is explained in the proof of Theorem 3. □

5.6 Greedy Range Search technique

We can now present a modified algorithm for a range search using pruning of the features. For a case of binary fingerprints, given a query q and a threshold t, the indexing algorithm goes as follows (As used earlier, N_q is the number of features in query q and V_i is the minimum number of features present among all points having the feature f_i.)

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. 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. This is because while

pruning this feature, if the point is present in some other feature (coming after feature f_1), then the point will become a candidate point later.

3. Hence if distance threshold is t and the following holds (as described in Equation 5.1)

$$\frac{1}{N_q - 1 + V_1} < 1 - t \tag{5.7}$$

we can prune the feature f_1

4. 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.

5. Hence if till the i^{th} feature is considered, if j features (call it set R) have been pruned till now, we can prune the i^{th} feature as well if the following holds (as described in Equation 5.5)

$$\frac{j + 1}{N_q - 1 + min(V_i, \rho)} < 1 - t \tag{5.8}$$

where ρ is the minimum number of features present in any point containing atleast one of the features pruned until now i.e $\rho = \min_{k \in R} V_k$

6. This process is continued till we reach the end of the sorted feature list.

7. We now follow the baseline range search technique described in section 5.4, but on the set of features of the query compound not pruned till now (neglecting the pruned features).

The main idea behind the use of the greedy algorithm is to prune as many as points as possible. Hence the sorting of the features in order of their popularity. We also tried sorting based on other scores but none gave as good as the results obtained by using popularity for sorting.

Consider the pruning process as described above. If say, we prune f_l, then we need to consider features until f_{l-1}, which we have pruned till now. Lets say, there is a point p_o, which has atleast one feature not seen till now , say $f_{l+q}, q \in (1, M - l)$. We do not care about such points, since they would be considered in the candidate set for out range search query if, in the future f_{l+q} is not pruned. We only care for those points which would not be present in any further features beyond feature f_l since we would be pruning them from our candidate sets.

5.7 Extension to non-binary fingerprints

We can extend Theorem 3 and Theorem 4 to non-binary fingerprints as follows:

Theorem 5. *For the case of non-binary fingerprints, given a query q and a threshold t, consider the feature f_i. If P is the set of points from the database which has non-zero f_i value, then we can prune*

all such points p_j of P from being present in the candidate range search set for query compound q if the point p_j has only f_i in common with the query q and it follows the following bound.

$$\frac{min(j_i, W_i)}{S_q - W_i - k_i + l_i + max(k_i, V_i)} < 1 - t \tag{5.9}$$

Here j_i is the maximum feature value taken for the feature f_i, W_i is the i^{th} feature value of query q, S_q is the sum magnitude of the feature values of the query q, k_i is the is the minimum feature value taken for the feature f_i, l_i is the minimum sum of feature values for any point containing the feature f_i and t is the threshold similarity.

Proof. This follows from the definition of the similarity measure for non-binary fingerprints as described in Equation 3.3. If the two compounds in question, q and p_j have only one feature f_i in common, then accordingly the maximum numerator value for the similarity score would be $min(j_i, W_i)$ while the denominator would take a minimum value of $S_q - W_i - k_i + l_i + max(k_i, V_i)$. The rest of the proof follows as in proof of Theorem 3.

□

Theorem 6. *For the case of non-binary fingerprints, given a query q and a threshold t, consider the feature set $F = f_1, f_2, ...f_M$. If P is the set of points from the database which has atleast one non-zero f_k value ($f_k \in F$), then we can prune all such points p_j of P from being present in the candidate range search set for query compound q if the point p_j does not have any other common feature other than in the set F, and it follows the following bound.*

$$\frac{\sum_{i \in J} min(j_i, W_i)}{S_q - \sum_{i \in J} W_i - \sum_{i \in J} k_i + max_{i \in J} l_i + \sum_{i \in J} max(k_i, V_i)} < 1 - t \tag{5.10}$$

Here J is the set of features already pruned, j_i is the maximum feature value taken for the feature f_i, W_i is the i^th feature value of query q, S_q is the sum magnitude of the feature values of the query q as mentioned earlier, k_i is the is the minimum feature value taken for the feature f_i , l_i is the minimum sum of feature values for any point containing the feature f_i and t is the threshold similarity

Proof. The proof follows from Equation 3.3 and proof of Theorem 4.

□

The algorithm in section 5.6 is to be modified as follows; for step 3 of the algorithm, while considering non-binary fingerprints we have to use the following inequality to determine if feature f_1 can be pruned.

$$\frac{min(j_1, W_1)}{S_q - W_1 - k_1 + l_1 + max(k_1, V_1)} < 1 - t \tag{5.11}$$

Similarly for step 5 of the algorithm in section 5.6: 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:

$$\frac{\sum_{i \in J} min(j_i, W_i)}{S_q - \sum_{i \in J} W_i - \sum_{i \in J} k_i + max_{i \in J} l_i + \sum_{i \in J} max(k_i, V_i)} < 1 - t \tag{5.12}$$

Remaining steps remain the same.

5.8 Using the Minimum Similarity Bound

Just like in the M-tree, similar to using the triangle inequality bounds, instead of using the maximum possible similarity, we used the minimum possible similarity and use it to include all points contained by a feature.

Hence for a feature f_i of a binary fingerprint query q, we can include all points having feature f_i in our result set if the following inequality holds:

$$\frac{1}{N_q - 1 + U_i} > 1 - t \tag{5.13}$$

Here N_q is the number of features in query q and U_i is the maximum number of features present among all points having the feature f_i. This holds true because the right hand side of the bound represents the least possible similarity between the query and any compound having feature f_i. If the left hand side is greater than the similarity threshold s ($s = 1 - t$), than we need not undergo the min-max similarity calculation for all the chemical fingerprints.

This is a very loose lower bound and hence we observed in our results that we were unable to exploit this bound effectively for speeding up our process.

5.9 Splitting Features

Another technique we considered for the binary data-set is the splitting of the heavy hitting features into two sets so that we can prune the bigger set (or the one containing the points which larger minimum number of features). If the bound, described in Theorem 4 for pruning the feature fails, then we try to split the feature into two sets so that the minimum number of features for any point in one of the sets increases and then giving a chance of the set being pruned. For this we have to sort the points in the feature in ascending order of the number of features present in the point. This procedure is applied only for the heavy hitting features since they appear in a large number of points. If we were to apply this to other features, the time spent partitioning would exceed the probable time spent in comparing all points in the partition to the query. Hence we would like to apply this only to features which are present in more than a threshold number of points.

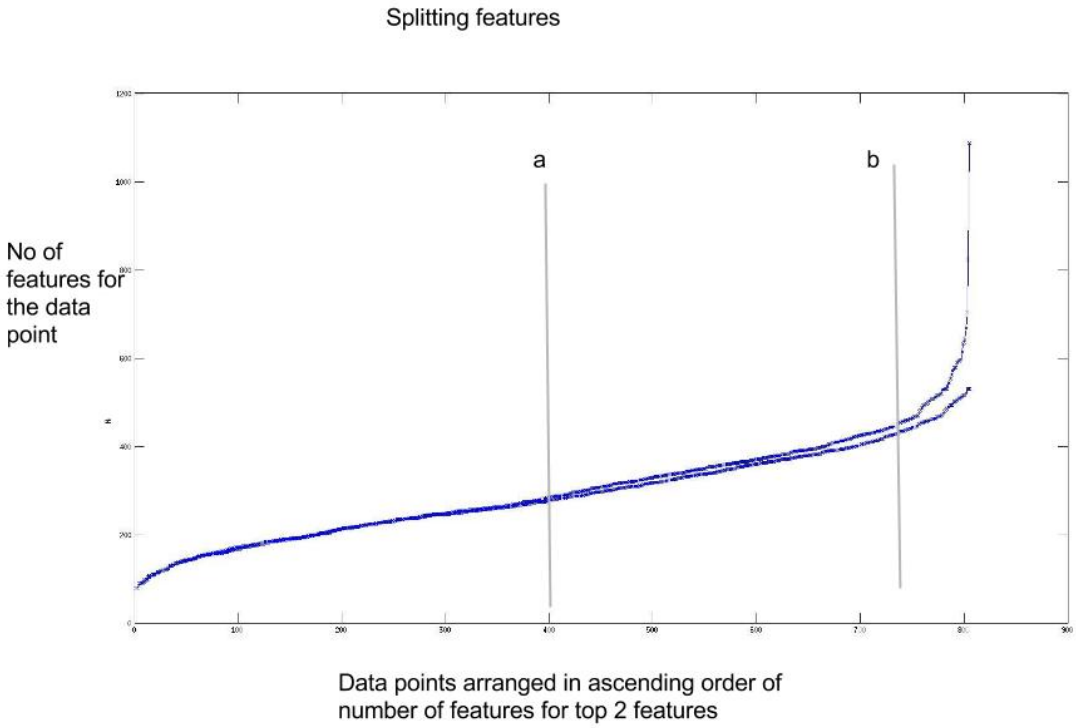


Figure 5.2: Inverted Indexing: Splitting the heavy hitting features

Figure 5.2 shows the number of features versus every point present for two heavy hitting features. It

can be observed that the curve is almost linear, hence intuitively we wouldn't get a very large increase in the minimum number of features for the larger set of points.In Figure 5.2, we can observe two lines *a* and *b*. If we were to split at *a* into approximately two equal sets) after sorting, our pruning bound would have a lower chance of getting satisfied while if we were to cut at *b* (at about 90% to 10%) we would have a better chance of pruning the set containing the top 10 percent of the points. Through experiments on the PubChem dataset (described later), we observed that we slowed down our range search process with this 'splitting features' method. We will not be using this method for the evaluations of our technique on various datasets.

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 behavior of the indexing techniques that we are applying, we perform further analysis of the data in the hope of getting some useful explanations. We want to ascertain that a typical fingerprint data-set is high dimensional and highly sparse almost following a power law sort of distribution with very few features occurring 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 Experiment Data-sets

The two real world data-sets we have explored in this work are from PubChem and DUD. PubChem is a public database of chemical compounds and a record of their activities and reactions against agents. It is maintained by the National Center for Biotechnology Information (NCBI), a component of the National Library of Medicine, which is part of the United States National Institutes of Health (NIH). We have taken a subset of compounds from their database for our experiments. We are using both binary and non-binary fingerprints which have been created using fingerprinting techniques applied on the chemical compound data from their databases. DUD is a directory of useful decoys for bench-marking 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 available compounds.

MOL2 is the primary molecule format used for the representation of chemical compounds. The MOL2 file displays information specifically about the following, Atom, Molecule, Bond, Substructure and Set. The atom record provides information about the atom names, types, coordinates, and partial charges. The molecule entry recognizes information about the molecule name and number of atoms, bonds, substructures and sets. Similarly the bond information displays the atom identifiers for every

bond and the substructure record identifies types of significant substructures seen.To extract fingerprints we have used MOLPRINT2D, a molecular fingerprinting technique.

We will be referring to the binary and non-binary versions of the PubChem data-sets as PubChem-b and PubChem-n respectively.

6.2 A Typical Fingerprint Data-set Analysis

We perform a detailed analysis of the data from PubChem-n 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: Data Analysis: Statistics of the data-set PubChem-n

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.

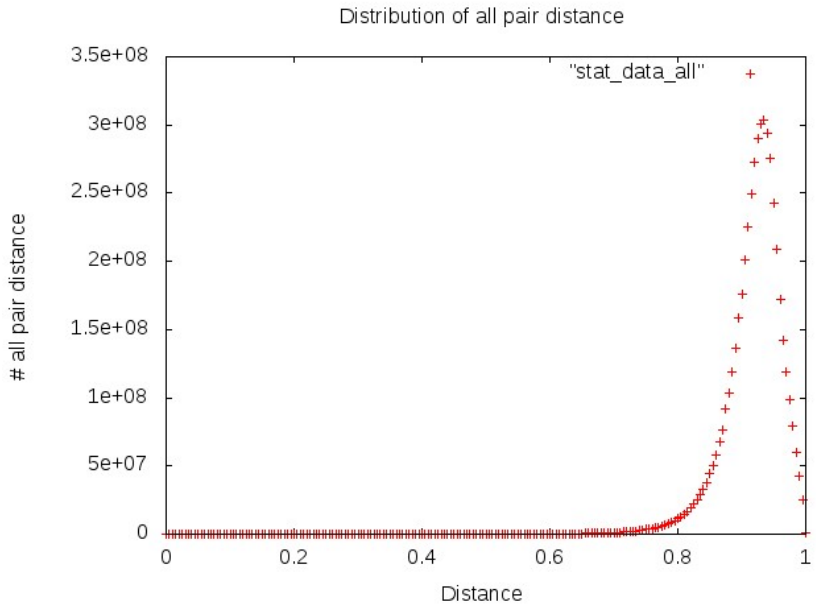


Figure 6.1: Data Analysis: All pair distances, plotted as number of pairs of points against the distance score between the points

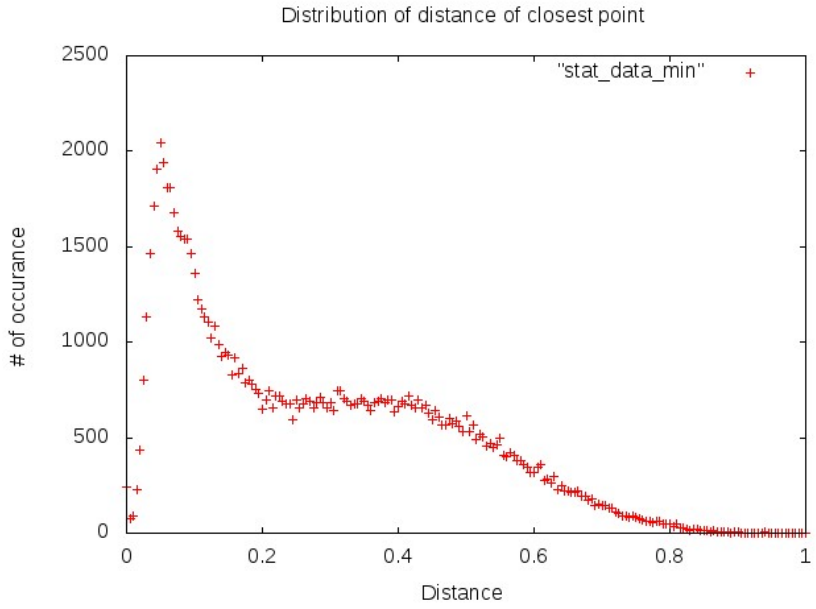


Figure 6.2: Data Analysis: Distance of the closest point, plotted as number of occurrences of the nearest neighbour distance against the distance score

6.3 Observations on the Data

We can observe from the data being in high dimension, that the points are spread out, and hence most of them are equidistant from each other. The closest point for most of the points is at distance much greater

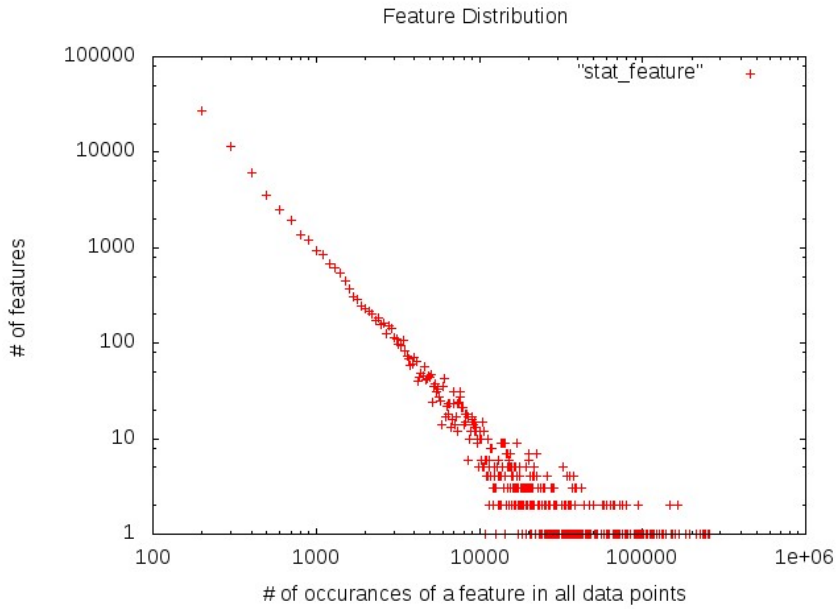


Figure 6.3: Data Analysis: Feature distribution, plotted as the number of features against the number of occurrences of the feature in all data points

than 0.4. in fact only 0.007% of the points have their 1-NN within a distance of 0.4.

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, this is commonly assumed for chemical data) i.e., even though we have many features only a hand full of them are repeated in most of the points.

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 could look into heuristic streaming algorithms to handle it efficiently (not explored in this work).

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 [13].

We observed that the baseline inverted index we proposed for point queries could not be easily generalized 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.

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 and Results

In our experiments, we evaluated our indexing techniques on two real world data-sets. We have compared our indexing techniques by comparing the running time with that of the state of the art technique. We are 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 data-set is the PubChem(-n and -b) data-set. As mentioned earlier, PubChem is a public database of chemical compounds maintained by the National Center for Biotechnology Information (NCBI), USA. The second data-set is from DUD, a directory of useful decoys for bench-marking 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 available compounds. To extract 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 empirically. We have used the PubChem-n data-set for all analysis of the M-tree. We use different data-set sizes of 1000, 10,000 and 100,000 number of chemical compounds.

For evaluation purposes, we have implemented a linear brute force scan to compute the range query and used that as a benchmark for the range search query 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 .

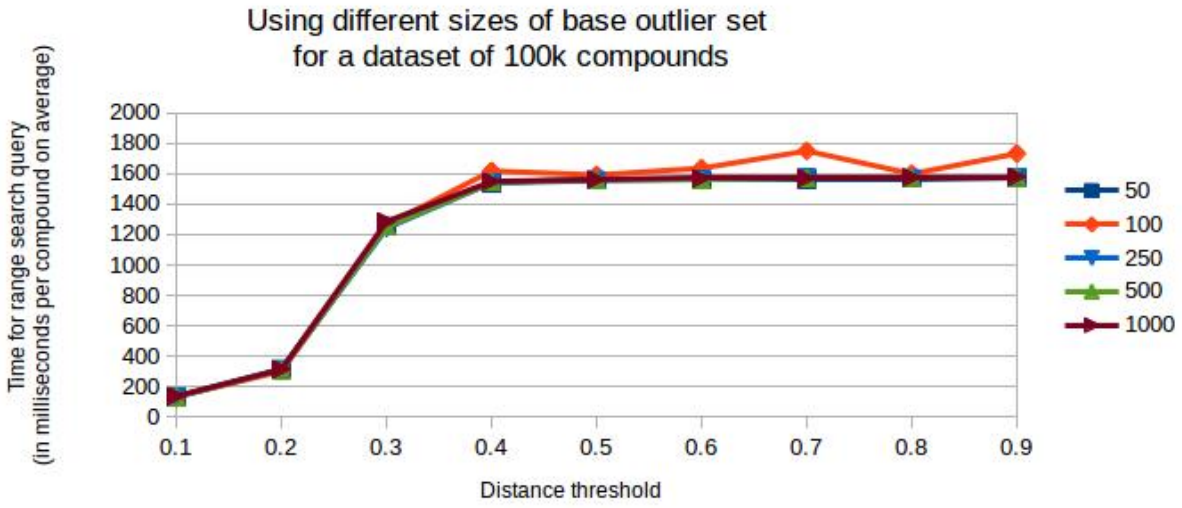


Figure 7.1: M-tree: Average Range Query time versus Distance Threshold for various sizes of base outlier set

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.

As seen in 7.1 we can observe that the range search query time is almost constant with change in size of limiting outlier set. 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 when a small 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 smaller base size limit for the outlier sets, the number of times the recursion is applied is high. 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 data-set size for all the future experiments. This parameter value has been decided empirically.

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 limiting size of the outlier set. 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.

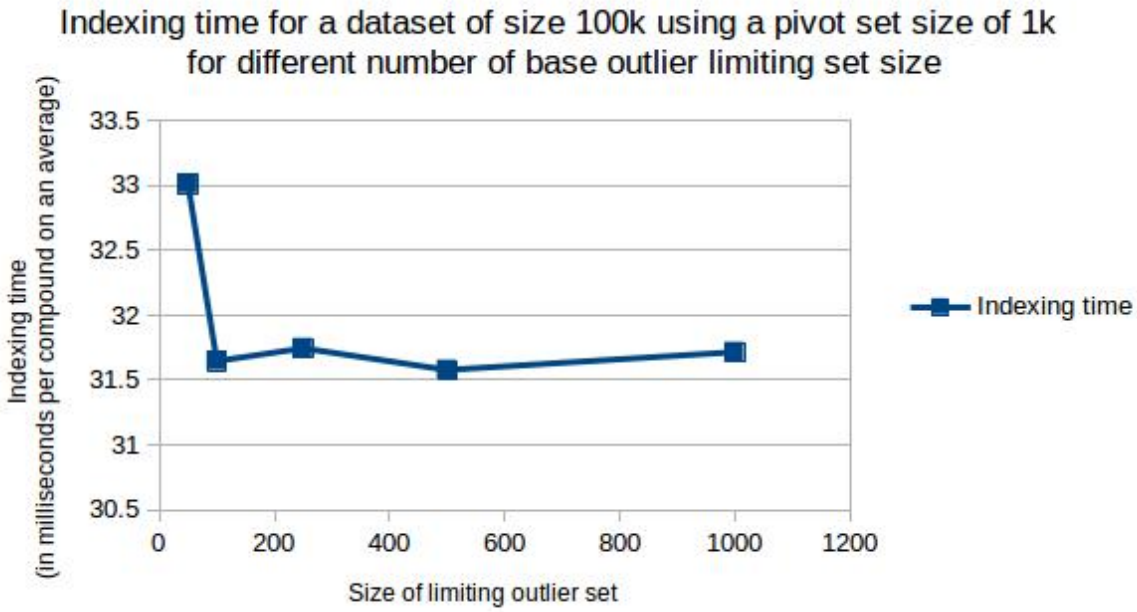


Figure 7.2: M-tree: Indexing time versus different sizes of base limiting outlier set

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). The parameter *p* determines the number of nodes at every level of the index structure. The number of nodes at each level is equal to p+1.

Figure 7.3, Figure 7.4 and 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 threshold values we are successfully 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. This is because the M-tree based index algorithm requires a re-indexing of all points as it becomes difficult to add individual points after the entire indexing process is over. Hence we need to have a threshold for indexing time.

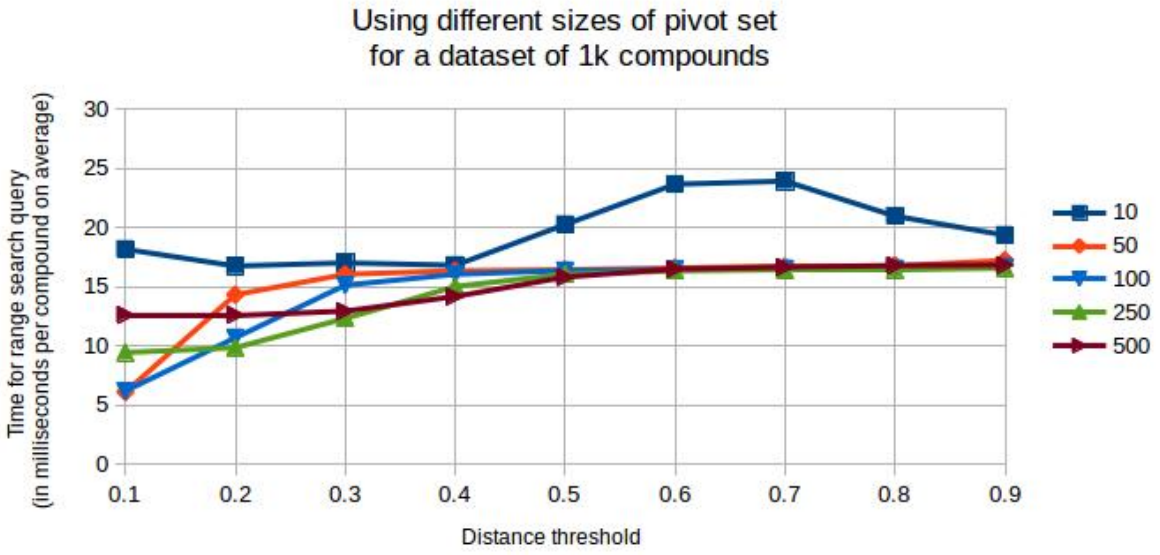


Figure 7.3: M-tree: Average Range Query time versus Distance Threshold for various sizes of pivot set - 1k PubChem-n database

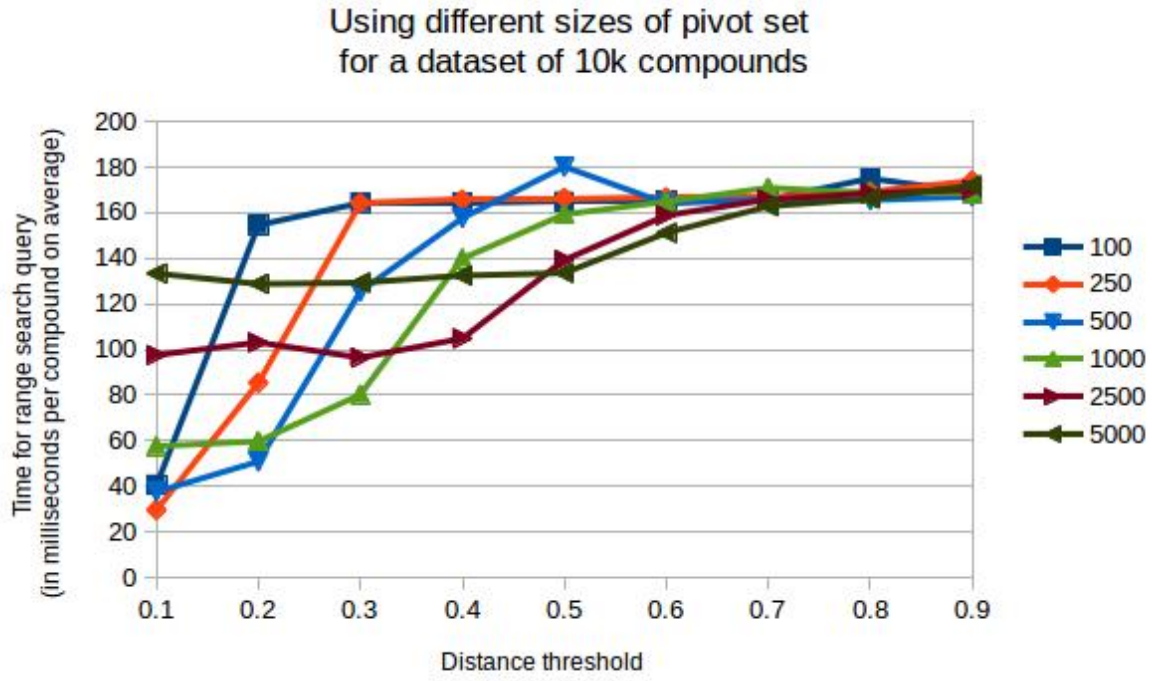


Figure 7.4: M-tree: Average Range Query time versus Distance Threshold for various sizes of pivot set - 10k PubChem-n database

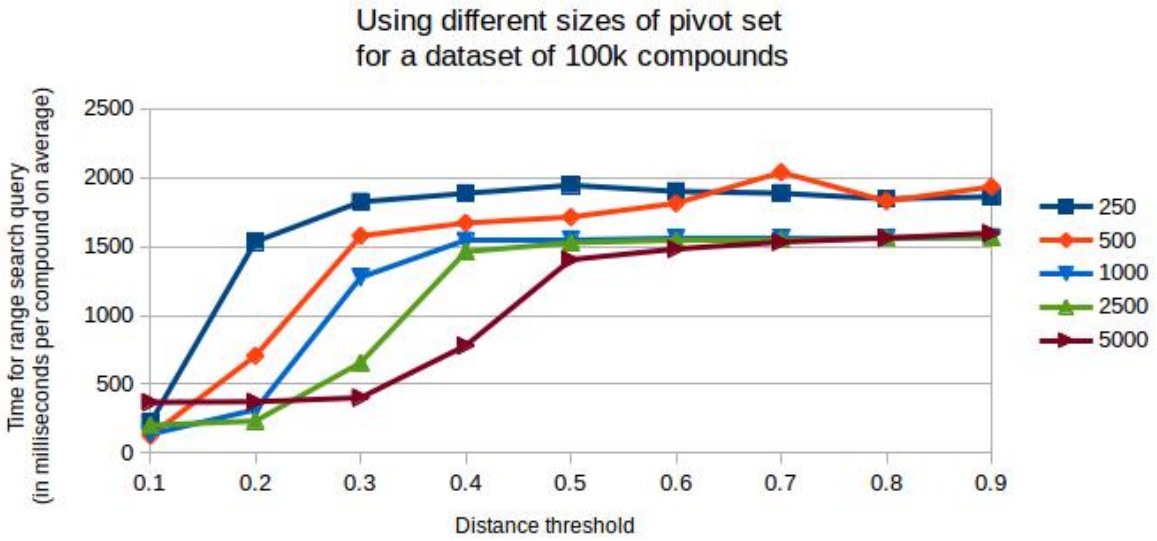


Figure 7.5: M-tree: Average Range Query time versus Distance Threshold for various sizes of pivot set - 100k PubChem-n database

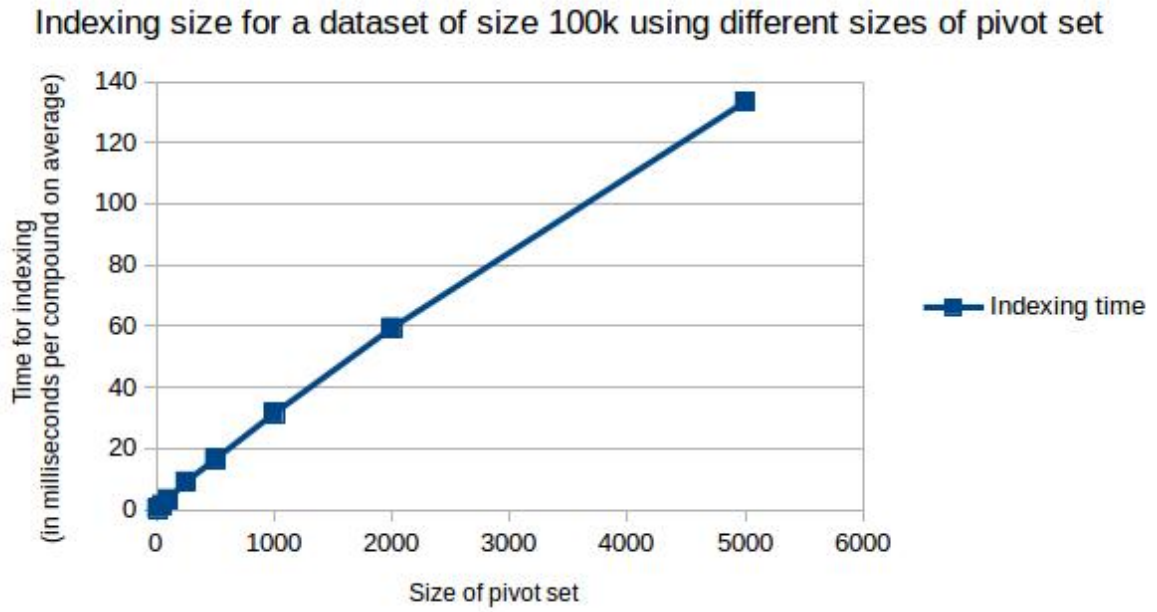


Figure 7.6: M-tree: Indexing time versus different sizes of pivot set size

7.1.3 Threshold Distance

As described in earlier sections, the range query distance threshold (t) is the cut-off 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 have

the number of pivots above a particular threshold.

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 small number of pivots, it is not possible to exploit the bounds effectively to decisively prune or include the sub-trees from our answer result set. For a small 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 Figure 7.7, we observe that for a 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 data-set size of 100k after which it increases. This is because there is a trade-off between the time saved by pruning the sub-trees 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 .

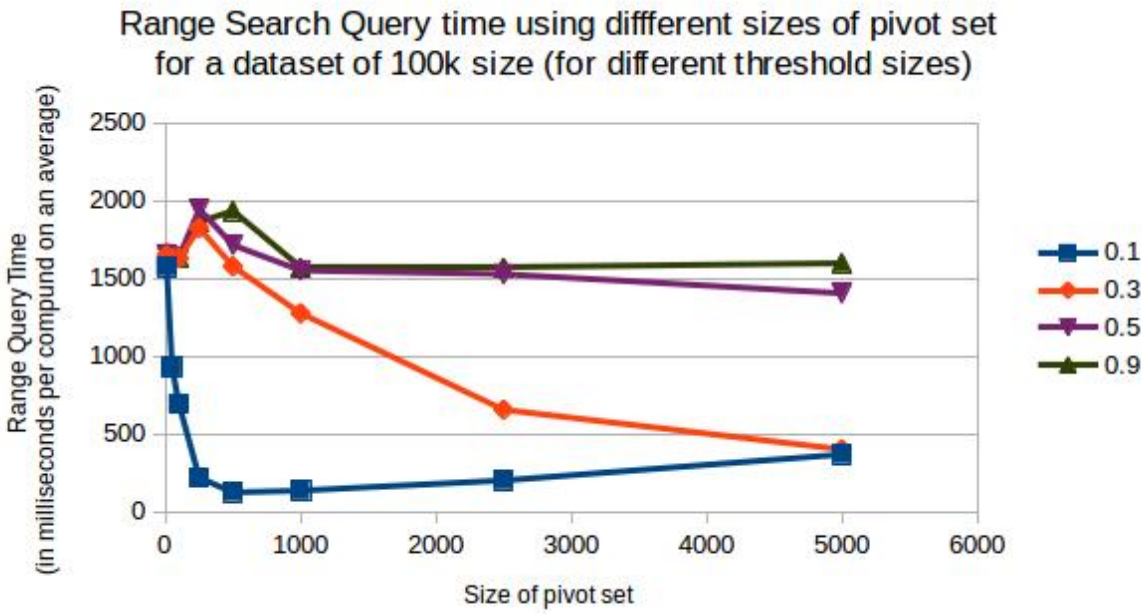


Figure 7.7: M-tree: Average Range Query time versus various sizes of pivot set for different threshold distances

7.1.4 Data-set Size

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

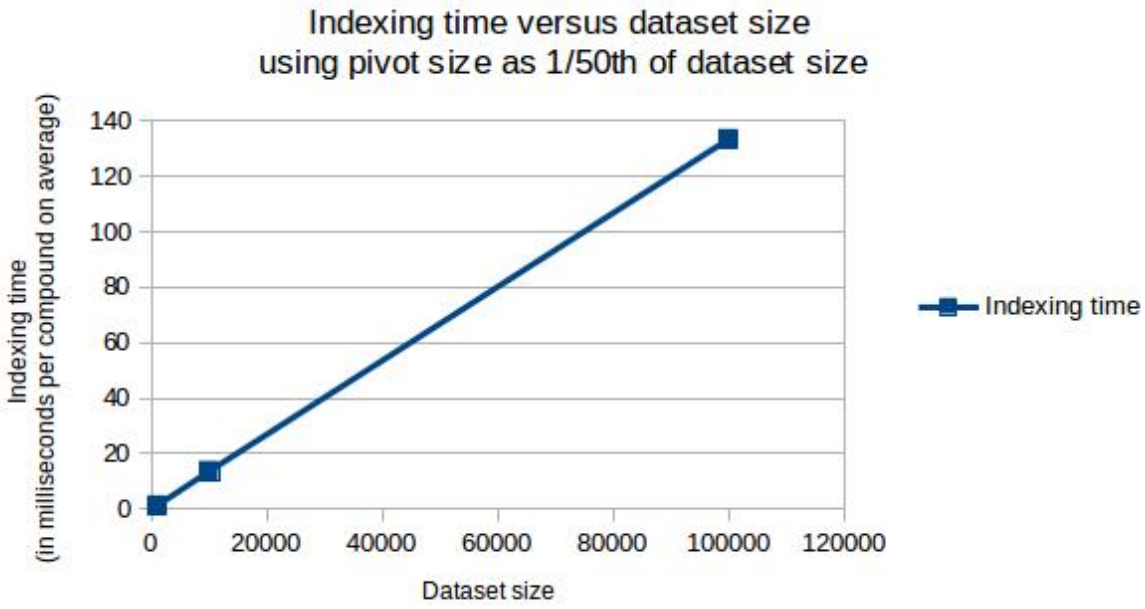


Figure 7.8: M-tree: Indexing time versus data-set size

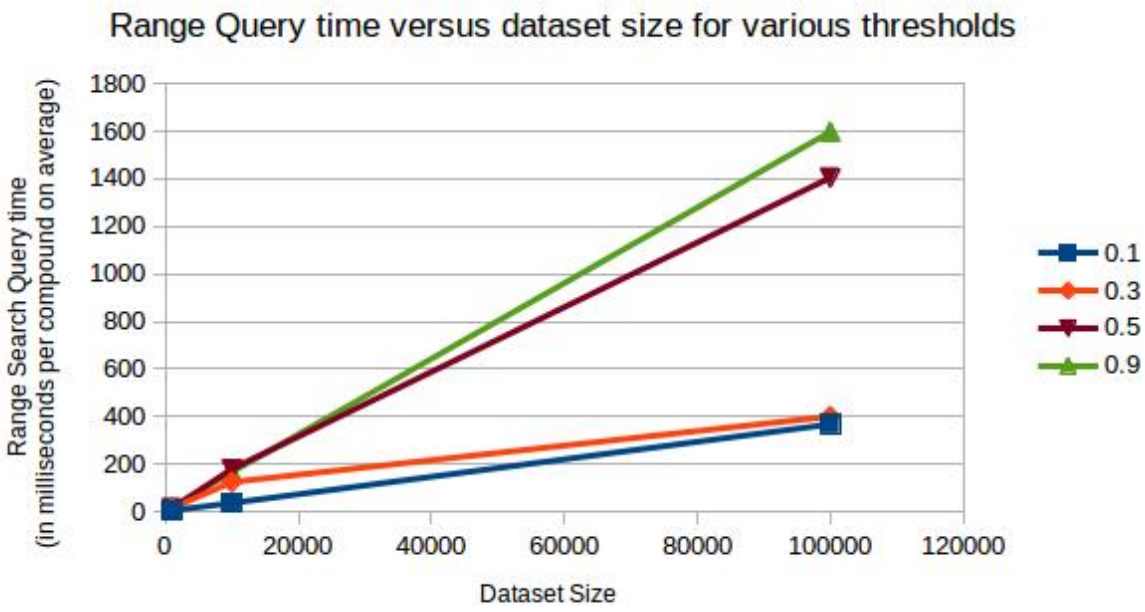


Figure 7.9: M-tree: Range Query time versus data-set size

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

As seen in Figure 7.8 we can observe that the average indexing time per compound on average

increases linearly with increase in data-set 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.

7.2 Inverted Indexing Analysis

For the below experiments, the test-bed 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 PubChem-n data-set. 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 Figure 7.10 the graph is almost straight parallel to the x-axis for both the binary as well as the non-binary data-set. This is in opposition to the M-tree index structure where the indexing time required per compound on average was increasing with increase in the data-set size.

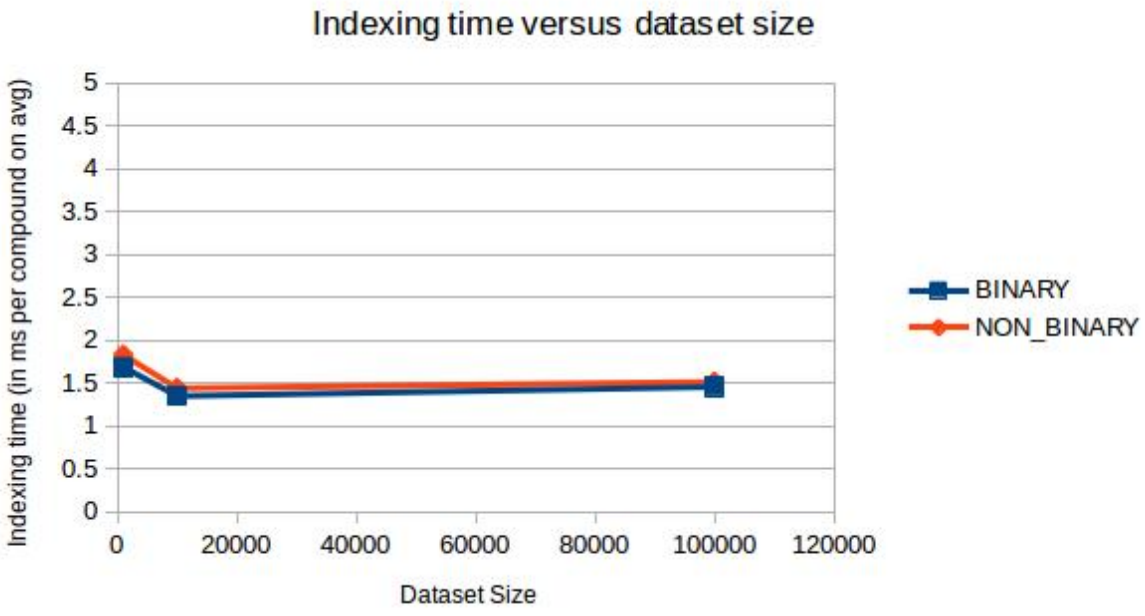


Figure 7.10: Inverted Index: Indexing time versus data-set size for Inverted Indexing

Figure 7.11 and Figure 7.12 show how the range query time graph increases with threshold distances

.We can see that in both the figures the time increases monotonically. The double derivative of the curve can be observed to be positive.

We observe that the range query time for the binary version of the PubChem-n data-set is lesser compared to the non-binary version of the same data-set. This is because of more pruning of features in the binary data-set.

Typically, in fingerprint data-sets, searching is done for a reasonable threshold distance of 0.1 to 0.3 and the search query times upto 0.3 distance is within a second per compound on average. The figures show that the indexing technique scales well on bigger data-sets for low threshold values, better than on higher threshold values.

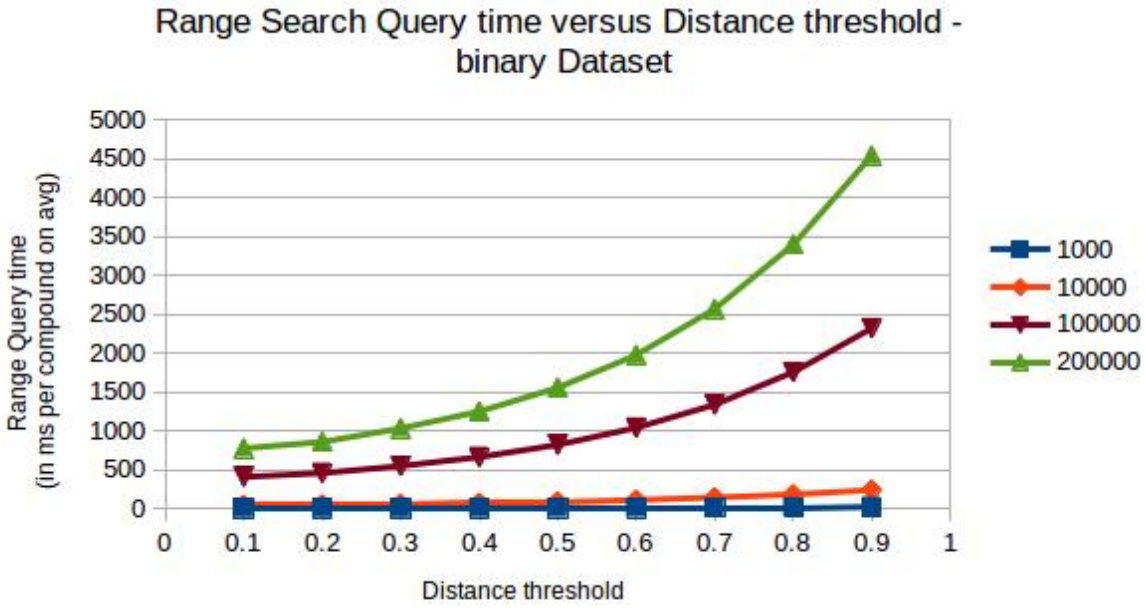


Figure 7.11: Inverted Index: Range Query time versus Distance threshold for different data-set sizes-PubChem-b data-set

We have experimented, with the mentioned improvisations in the Inverted Indexing technique, of splitting the heavy hitting features as described in section 5.9. We observed an increase of about 50-100 ms per compound for the splitting technique.

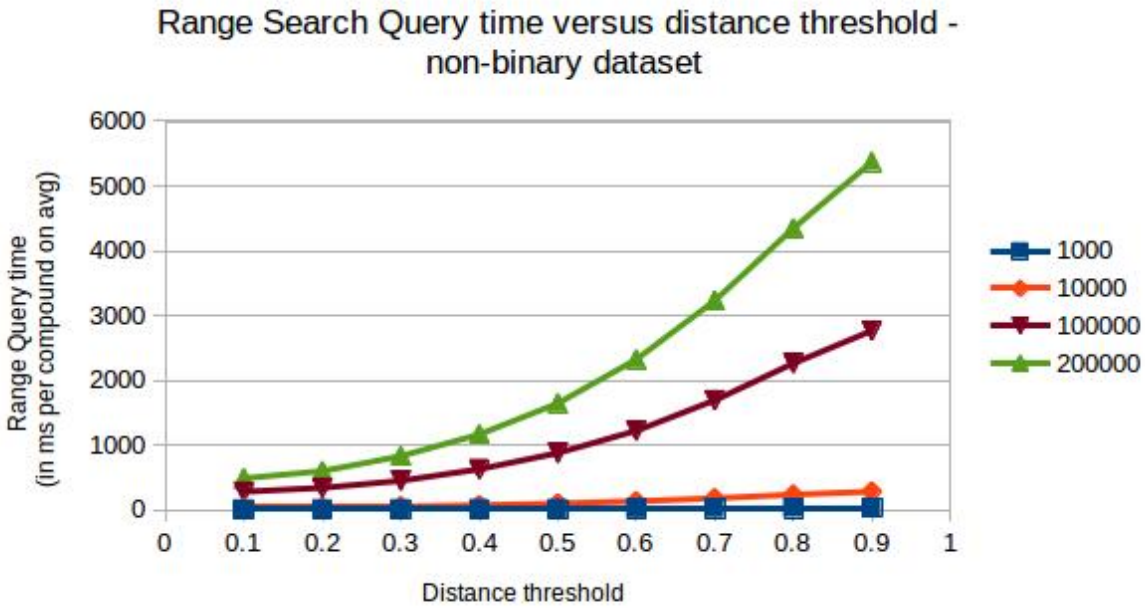


Figure 7.12: Inverted Index: Range Query time versus Distance threshold for different data-set sizes- PubChem-n data-set

7.3 Comparison of our techniques

We have compared the three techniques, namely the M-tree indexing technique , the Inverted Indexing technique and the bit bounding technique described in Swamidass and Baldi [17]. We have used the PubChem-n, PubChem-b and DUD data-sets for our evaluations. A test set of 500 compounds were chosen randomly from the data-sets for evaluation purposes. For the experiments, the test-bed was a server with 512GB RAM, 24TB disk space, and 2 quad core Xeon processor.

As observed in Table 7.1, a comparison of the techniques on the PubChem-b data-set shows us that for a very low threshold value, the M-tree index showed the most promising results, achieving upto two times the speed of the bit bound technique. The inverted index is also able to achieve 2-3 times speed-up compared to the bit bound technique.

We have shown the effectiveness of out technique on non-binary data-sets, a feature unexplored in most papers in the field of research. We can observe from the Table 7.1 that for high values of distance threshold all three techniques start converging to similar run-time for the range search query.

Distance threshold	Mtree	Bit bound	Inverted Indexing
0.1	386.286549	820.3638749	485.470212
0.2	550.1953386	1661.947007	596.3910451
0.3	2473.023198	2524.738663	826.6685434
0.4	4936.808636	3363.817437	1162.332165
0.5	5063.290607	4090.751155	1643.795028
0.6	5122.609976	4654.190322	2321.426377
0.7	5145.041938	5018.220471	3230.933406
0.8	5152.297697	5175.073409	4346.756728
0.9	5178.022697	5229.718132	5371.661469

Table 7.1: Comparison of techniques applied on the PubChem-n fingerprint data-set

Distance threshold	Mtree	Bit bound	Inverted Indexing
0.1	384.8059272	708.6588827	776.7649653
0.2	480.429907	1444.478385	859.3060838
0.3	1941.113211	2217.534745	1033.699747
0.4	4896.263139	2995.191962	1251.065067
0.5	5037.030643	3736.857438	1556.630309
0.6	5086.700389	4354.433042	1976.196625
0.7	5118.526736	4859.848535	2568.638003
0.8	5138.21594	5082.24385	3396.452903
0.9	5168.943088	5138.640137	4540.309089

Table 7.2: Comparison of techniques applied on the PubChem-b fingerprint data-set

Distance threshold	Mtree	Bit bound	Inverted Indexing
0.1	107.1823181	146.6163294	29.37458224
0.2	297.8381414	217.9150917	35.10997253
0.3	303.2994695	272.7101361	39.32821769
0.4	295.2179914	292.4566623	43.20251988
0.5	299.248134	304.6924106	56.92043218
0.6	297.995192	294.1471833	69.95566644
0.7	289.2738744	295.0616706	91.30182238
0.8	301.1660971	287.0035369	127.5203695
0.9	294.3855533	289.8519661	191.9871398

Table 7.3: Comparison of techniques applied on the binary DUD data-set

Table 7.2 shows the comparison of techniques applied on the PubChem-b data-set while Table 7.3 shows the comparison of the techniques on the binary DUD data-set. The performance of M-tree index can be seen to be comparable to the Bit-bound technique while the performance of the inverted index can be observed to be superior than both the other techniques. For the PubChem-b data-set, we observe that there is always a 2-2.5 times speedup for the inverted index as compared to the bit bound technique. The M-tree index beats the other two techniques for low threshold values but there is a sharp increase in range search query time around 0.3 where the run-time shoots up.

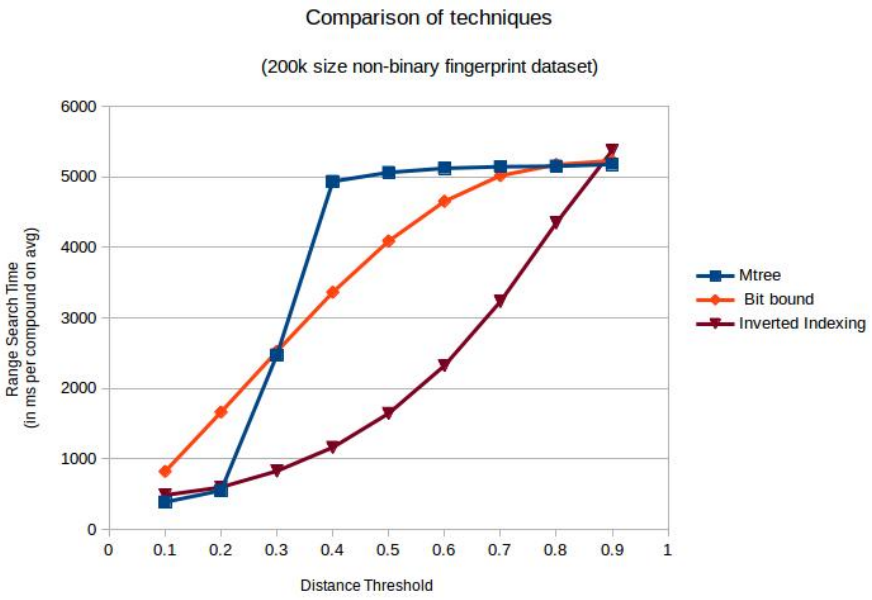


Figure 7.13: Comparison of techniques applied on the PubChem-n fingerprint data-set

Comparison on the DUD data-set showed that the inverted indexing technique was able to achieve 5-6 times speedup on the M-tree index and the bit bound algorithm. As observed in Figure 7.13, Figure 7.14 and Figure 7.15 our inverted indexing technique works better than the other two on an average. The M-tree technique works well for lower threshold values, sometimes better than the inverted index technique but fails to scale for higher threshold values.

Comparing the M-tree index structure with inverted index technique, we observe the following: The indexing time per compound on average is constant for the inverted index structure while it varies linearly with size of the data-set for the M-tree. The run-times for range search queries tends to converge for both the methods as we go towards higher threshold values. The M-tree index shows a sudden increase in run-time for range query at around 0.3 while the curve for the inverted index is smoother. Also, if we were to include a new point into the chemical database the inverted index structure would be simpler since we would just need to update the corresponding feature lists. But for the M-tree index, the indexing process would need to be repeated from start for better compact clusters, since it would be difficult to add the point randomly into the tree structure.

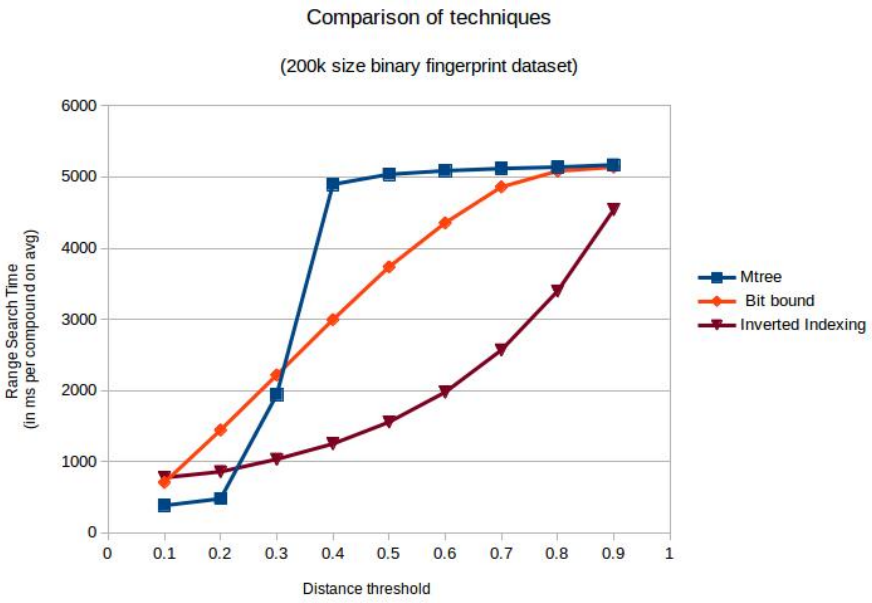


Figure 7.14: Comparison of techniques applied on the PubChem-b fingerprint data-set

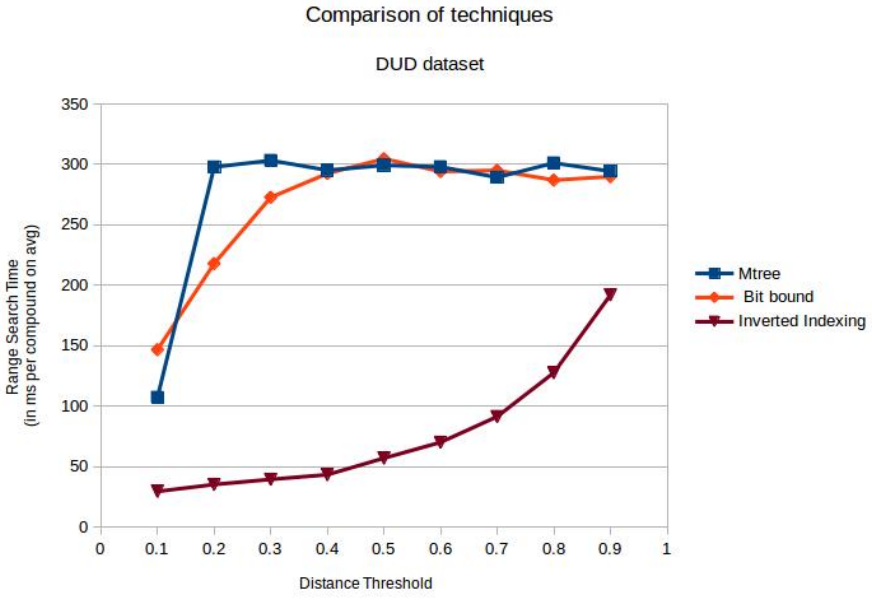


Figure 7.15: Comparison of techniques applied on the binary DUD data-set

CHAPTER 8

Conclusion and Future Work

In this work, we have proposed two different indexing structures, namely an M-tree based index structure and an inverted index structure for the task of range search over a database of chemical compounds. We have discussed methods to carry out a range search process for each of the two methods and have outlined the indexing as well as the search algorithm in detail. Through experiments on real world data-sets, we have been able to show the effectiveness of our algorithms through comparison with the state of the art *Bit Bounding* technique.

The first indexing structure we have described in this work is an M-tree based index, capable of range search queries on both binary and non-binary chemical fingerprints. We have proposed an unique indexing procedure, which was able to handle the outliers efficiently. The indexing technique exploited the metric property of the Tanimoto and Min-Max distance measures as well as the structure of the M-tree. We analyzed a fast pivoting method which was able to achieve a linear time (per compound on average) indexing run-time. We have undertaken a detailed analysis of the M-tree based index through experiments on real world data-sets. We have successfully achieved a 2-fold improvement in query time compared to the Bit bound technique. In addition, we showed that the contractive mapping of Lipschitz Embedding was unable to fasten the range search query process.

Motivated by the use of inverted index in text mining and information retrieval, we explored an inverted index structure for the range search query process. The indexing process of the inverted index takes very minimal time. We proposed a novel indexing technique which relied on the pruning of features i.e. whose points were guaranteed to be absent in the result set. This pruning was achieved using bounds which we have derived in section 5.5 and section 5.7. We achieved upto 6-times speed-up compared to the Bit bound technique. We have shown through experiments the effectiveness of our algorithms on non-binary data-sets which no previous work in this field have shown.

Future extensions to our work can focus on methods to facilitate a top-k search in addition to the

range search for our index structures. We can try to adapt our structures for the fast computing of the nearest neighbor for all the points in the database. The range search technique, we have proposed for the inverted index is a greedy algorithm. We can look into other heuristics which could help prune the feature set more effectively.

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] **Bolton, E. E.**, **Y. Wang**, **P. A. Thiessen**, and **S. H. Bryant** (2008). Pubchem: integrated platform of small molecules and biological activities. *Annual reports in computational chemistry*, **4**, 217–241.

[6] **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.

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

[8] **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.

[9] **Huang, N.**, **B. K. Shoichet**, and **J. J. Irwin** (2006). Benchmarking sets for molecular docking. *Journal of medicinal chemistry*, **49**(23), 6789–6801.

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

[11] **Klein, K.**, **N. Kriege**, and **P. Mutzel**, Ct-index: Fingerprint-based graph indexing combining cycles and trees. *In Data Engineering (ICDE), 2011 IEEE 27th International Conference on*. IEEE, 2011.

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

[13] **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.

[14] **Napolitano, F.**, **R. Tagliaferri**, and **P. Baldi**, An adaptive reference point approach to efficiently search large chemical databases. *In Recent Advances of Neural Network Models and Applications*. Springer, 2014, 63–74.

[15] **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.

[16] **Nasr, R.**, **R. Vernica**, **C. Li**, and **P. Baldi** (2012). Speeding up chemical searches using the inverted index: The convergence of chemoinformatics and text search methods. *Journal of chemical information and modeling*, **52**(4), 891–900.

[17] **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.

[18] **Thiel, P.**, **L. Sach-Peltason**, **C. Ottmann**, and **O. Kohlbacher** (2014). Blocked inverted indices for exact clustering of large chemical spaces. *Journal of chemical information and modeling*, **54**(9), 2395–2401.

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