DATA SENSITIVE APPROXIMATE QUERY APPROACHES IN METRIC SPACES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Merve Dilek

September, 2011

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. İbrahim Körpeoğlu(Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Defne Aktaş

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Dr. Cengiz Çelik

Approved for the Graduate School of Engineering and Science:

Prof. Dr. Levent Onural
Director of Graduate School of Engineering and Science

ABSTRACT

DATA SENSITIVE APPROXIMATE QUERY APPROACHES IN METRIC SPACES

Merve Dilek
M.S. in Computer Engineering
Supervisor: Assoc. Prof. Dr. İbrahim Körpeoğlu
September, 2011

Similarity searching is the task of retrieval of relevant information from datasets. We are particularly interested in datasets that contain complex and unstructured data such as images, videos, audio recordings, protein and DNA sequences. The relevant information is typically defined using one of two common query types: a range query involves retrieval of all the objects within a specified distance to the query object; whereas a k-nearest neighbor query deals with obtaining k closest database objects to the query object. A variety of index structures based on the notion of *metric spaces* have been offered to process these two query types.

The query performances of the proposed index structures have not been satisfactory particularly for high dimensional datasets. As a solution, various approximate similarity search methods offering the users a quality/time trade-off have been proposed. The rationale is that the users might be willing to tolerate query precision to retrieve query results relatively faster. The proposed approximate searching schemes usually have strong connections to the underlying data structures, making the comparison of the quality of the essence of their ideas difficult.

In this thesis we investigate various approximation approaches to decrease the response time of similarity queries. These approaches use a variety of statistics about the dataset in order to obtain dynamic (at the time of querying) and specific guidance on the approximation for each query object individually. The experiments are performed on top of a simple underlying pivot-based index structure to minimize the effects of the index to our approximation schemes. The results show that it is possible to improve the performance/precision of the approximation based on data and query object sensitive guidance.

Keywords: Approximate Similarity Searching, Metric Spaces, Range Query.

ÖZET

METRİK UZAYLARDA VERİ DUYARLI YAKLAŞIK SORGULAMA YÖNTEMLERİ

Merve Dilek
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Doçent Dr. İbrahim Körpeoğlu
Eylül, 2011

Benzerlik taraması veri kümelerinden ilgili bilginin elde edilmesi işlemidir. İlgi dahilindeki veri kümeleri özellikle resim, görüntü, ses kaydı, protein ve DNA dizisi gibi karmaşık ve düzensiz veriler içerirler. İlgilenilen bilgi genellikle iki yaygın sorgu türünden bir tanesi kullanılarak tanımlanır: Menzil sorgusu, verilen sorgu nesnesinin belirli bir uzaklığı içerisinde kalan bütün nesnelerin elde edilmesini kapsar. Öte yandan en yakın k komşu sorgusu, sorgu nesnesine en yakın k veritabanı nesnesinin elde edilmesi ile ilgilenir. Belirtilen sorgu türlerini uygulayabilmek amacıyla *metrik uzay* kavramına dayanan çeşitli indeks yapıları önerilmiştir.

Önerilen bu indeks yapılarının sorgu performansları özellikle yüksek boyutlu veri kümeleri için çok tatmin edici olmamıştır. Çözüm olarak, kullanıcılara kalite/zaman ödünleşim imkanı sunan çeşitli yaklaşık benzerlik taraması yöntemleri geliştirilmiştir. Bu yaklaşım, kullanıcıların sorgu doğruluğundan ödün vererek sorgu sonuçlarına görece daha hızlı erişmek istemeleri ilkesine dayanmaktadır. Önerilen yaklaşık tarama tasarıları genelde altta kullanılan veri yapılarına çok bağımlıdırlar. Bu durum, bu tasarıların dayandığı temel fikirlerin kalite açısından kıyaslanabilmesini zorlaştırmaktadır.

Bu tezde, benzerlik sorgularının cevap süresini kısaltabilmek için farklı yaklaşık benzerlik yöntemleri araştırılmıştır. Bu yöntemler, elimizdeki veri kümesinden elde edilen çeşitli istatistiksel bilgileri kullanarak her sorgu nesnesine özgü dinamik (sorgulama esnasında gerçekleşen) yönlendirmeye olanak sağlamaktadırlar. Deneyler basit bir pivot-tabanlı indeks yapısı üzerinde çalıştırılarak alttaki yapının yaklaşık benzerlik tasarılarına etkisi azaltılmıştır. Sonuçlar, veri kümesine ve sorgu nesnesine duyarlı yönlendirmenin performans/doğruluk hususunda iyileştirme sağlayabileceğini göstermektedir.

Anahtar sözcükler: Yaklaşık Benzerlik Taraması, Metrik Uzaylar, Menzil Sorgulama.

Acknowledgement

I would like to express my gratitude to Dr. Cengiz Çelik for his support, encouragement and inspiration throughout this thesis.

I would like to thank to my supervisor Assoc. Prof. Dr. İbrahim Körpeoğlu for his support and helps and Asst. Prof. Dr. Defne Aktaş for accepting to read and review the thesis.

I would like to thank my husband Alptuğ for his understanding, support, and love at all time. He always encouraged me and made me smile with his existence.

Finally, I would like to thank my parents Yasemin and Mustafa, and my sister Deniz for their support during my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Searching is one of the fundamental problems in computer science [12]. In traditional way of searching, one is generally interested in *exact* searching, where objects satisfying a given search criteria exactly are returned as the result set. Exact searching is mostly encountered in traditional databases, which contain structured data. However, the increase in storage and availability of complex and unstructured data resulted in a necessity to perform a different type of searching: *Similarity Searching.*

In similarity searching, the intent is to return not exactly identical, but somewhat close objects to a given query object. Since the data has no structure, the only applicable search criteria is an object from the same domain. The similarity searching is applicable in a wide range of contemporary database and applications such as images [18], text files [3], videos, audio recordings, DNA and protein sequences [30], fingerprints [22], face recognition [21], etc. The data available in such formats are complex and unstructured, hence it is not possible to store them in traditional databases. A popular approach is to represent data as feature vectors and define the similarity between these objects in regards to the geometric distance between the vectors. Such an approach suffers heavily when the vectors are high-dimensional. Also the search is not performed on the real objects, but only on what can be captured by the feature vectors. A more general solution to the similarity searching is based on "metric spaces".

1.1 Metric Space

A metric space is composed of a universe of objects X and a distance function d defined between the objects of that universe. The distance function satisfies all the following properties for every a, b, and c in X universe.

- Positivity: $d(a, b) \geq 0$
- Symmetry: $d(a, b) = d(b, a)$
- Reflexivity: $d(a, a) = 0$
- Triangular Inequality: $d(a, b) + d(a, c) \geq d(b, c)$

The metric space definition given above is a generalization of the vector spaces where no assumption about the underlying structure of the data is made. In other words, it is not necessary to represent the items of the database as vector spaces, all that needed is a distance function satisfying the properties mentioned above.

It is the triangular inequality property of the distance function that is useful in deciding the similarity of objects. Nearly all of the index structures built on metric spaces store distances between all the database objects and a very small subset of special objects (called pivots or vantage points). With the distances between a pivot and many database objects at hand, one can make estimation about the distance between a query object and all the database objects by just calculating the distance between the pivot and the query object.

Let the pivot be represented by p, query object by q, and database object by o. The distance between p and o, dpo, is already stored and the distance between p and q, dpq, is calculated. By using the triangular inequality, the following two can be derived:

- $dqo \geq |dpq - dpo|$
- $dqo \leq dpq + dpo$

This means that the distance between the query object and the database object has a calculated lower and upper bound. These bounds are used in answering two different types of similarity queries faster: Range Query and K-Nearest Neighbor (k-nn) Query.

1.2 Similarity Queries

In a range query, the user supplies a query object of interest, q, and a radius value of r to a dataset X. The result set to be returned to the user contains all the database objects residing in r range of the query object. A range query can be summarized more formally as follows: $Range(q, r) = o \in X : d(q, o) \leq$ r

The figure below illustrates the visualization of a range query. All the objects displayed with a green dot are returned to the user as the result of the range query.

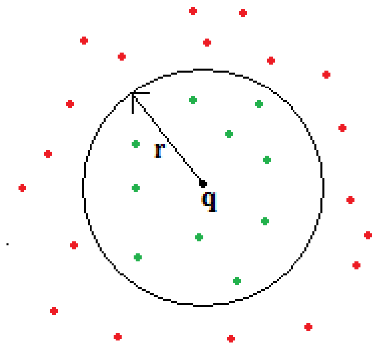


Figure 1.1: Visualization of the range query for query object o and radius value of r.

In a metric space, if the lower bound of the distance between the query object and a particular database object is greater than query radius it is obvious that *dqo* (the distance between query object and the database object) is greater than radius as well. This means the database object is outside the query range of the query object. Conversely if the radius is greater than the upper bound, it is clear that *dqo* is less than radius. This means that the particular database object is in query range of the query object. The database objects for which a decision is made by either way is accepted to be eliminated (the calculation of the actual distance dqo is not necessary). The actual distance calculation is needed to be performed if elimination is not possible.

In a $k - nn$ query, user-supplied radius value in the range query is replaced with the number of elements to be retrieved in the result set. In other words, the user provides a query object for which he/she desires to obtain k closest objects from dataset X as the result. Formal representation of a $k - nn$ query is: $Knn(q, k) = R \subseteq X,\ o1 \in R,\ o2 \in X - R,\ |R| = k : d(q, o1) \leq d(q, o2)$

The figure below illustrates the visualization of a $k - nn$ query, where the database objects tagged with n1-n4 are returned to the user as the result of the $k - nn$ query.

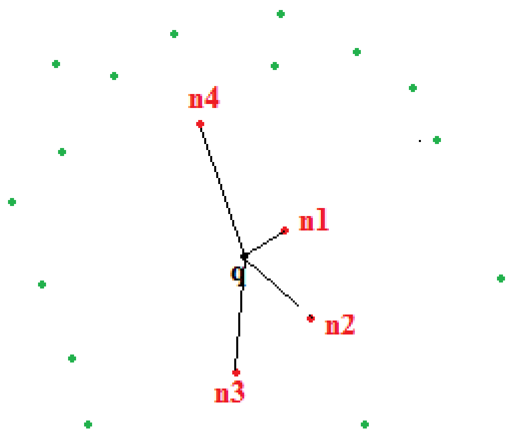


Figure 1.2: Visualization of the $k - nn$ query for query objet o and k value of 4.

1.3 Motivation

The users might want to retrieve the results of both query types faster while relaxing the correctness or integrity constraints of the result. In other words, they might want to retrieve only a subset of all the correct results even with addition of few erroneous objects. Such a motivation attracted the interests of many researchers and different approximate similarity search algorithms have been offered. Most of these algorithms are heavily dependent on the underlying index structure and provide improvements specialized for that structure. Our motivation is to show that approximate similarity searching can be performed via simpler methods not specifically designed for a particular index structure.

1.4 Contributions

With the motivation mentioned above, as opposed to restricting our interest for the improvement of approximation algorithms developed for a specific index structure, we showed that information derived from a particular dataset of interest might be helpful in deciding on similarity search for a specific query object. We named such an approach as "Data Sensitive", since dynamic guidance for the query object is provided with respect to the information gathered from the dataset and the query object itself.

1.5 Organization of the Thesis

The organization of the rest of this thesis is as follows: In Chapter 2, a brief survey of the existing work on similarity searching, and approximate similarity searching along with categorization of the latter is provided. Chapter 3 is composed of various methods and techniques that are used commonly by different approximation algorithms we propose. Chapter 4 is dedicated to the algorithms proposed, in which the details and rationale of the algorithms are provided. Chapter 5 contains results of the experiments performed aside from the evaluation of the proposed algorithms. Finally conclusion and future work are explained in Chapter 6.

Chapter 2

Background and Related Work

Exact match is the retrieval of records that are exactly same with the query object; whereas similarity search is the retrieval of similar records to the query object as mentioned in Chapter 1. For many applications similarity search is preferable over exact match. In this chapter, initially we introduce different index structures developed for similarity searching based on metric spaces along with a simple categorization of them. The motivation behind approximate similarity searching, which provides improvements on the query performance of similarity searching via decreasing the correctness of the results, is discussed and explained shortly. The chapter concludes with the classification of existing approximate similarity searching approaches into three categories.

2.1 Index Structures

We can categorize indexing methods into two groups as *clustering based* and *pivot based* methods. In clustering-based methods, the space is partitioned into clusters, and cluster centers are used to represent these partitions. Using this cluster centers' distances to the query object, queries may eliminate regions based on triangular inequality. *Generalized-Hyperplane Tree* (GHT) [29], *Geometric Near-neighbor Access Tree* (GNAT) [6], *M-Tree* [16], *Slim-Tree* [28], *vp-Tree* [29], and *mvp-Tree* [5] are among the most important clustering-based methods.

Pivot-based methods use a subset of the objects which are called pivots, and an index stores the distances between the objects and the pivots. These distances are used to eliminate some objects by using the triangular inequality. *Approximating and Eliminating Search Algorithm* (AESA) [26], *Linear AESA* (LAESA) [23], *Fixed-Queries Array* (FQA) [10] , *Spaghettis* [9] and *KVP Structure* [8] are well known pivot-based methods.

The evaluation of these various indexing methods is performed according to their query performance. The number of distance calculations is the main issue that measures the cost of the query. *Computational overhead* and *CPU overhead* terms are also used as additional computation costs.

2.1.1 Clustering Based Methods

The main principle of Clustering-Based Method is the hierarchical decomposition of the data space. One of the approaches that Tree-Based Methods use is based on grouping close objects in sub-trees. An object, which exists near the center of the groups, is selected as a representative of these sub-trees. Some Tree-Based methods use local pivot approach that is based on using one or more local pivots selected from the database. Partitioning is performed by using the distance information of objects to local pivots. This time subtrees contain objects with similar distances to the selected pivot(s).

GHT uses the hyperplane between two representatives selected from the subset. The remaining of the subset is partitioned into two according to their closeness to these representatives. GNAT generalizes GHT by using more than two representatives that are used in partitioning. Each internal node of the tree stores $m \times m$ table, where m is the number of clusters. The cells in the table contain information of minimum and the maximum distances between cluster centers to the objects in other clusters. Cluster elimination is performed according to the values in the cells of the tables.

M-Tree is a disk-based structure which is efficient at performing queries. It allows split and merge operations and still optimizes the IO performance. It stores the maximum distance to objects in a subtree. Overflow cases can be handled by performing node splitting. Node splitting is done by selecting two pivots and distributing the objects among two pivots. In order to have the tightest covering radius, M-Tree tries every possible situation and chooses the one which has the tightest covering radius. Slim-Tree improves M-Tree by introducing a more efficient splitting approach, while keeping the same structure with M-Tree. In this approach minimum spanning tree of the objects is generated. Slim-Tree also performs split and merge operations efficiently but in terms of query performance, it is less preferable over GNAT structure.

Vp-Tree is a tree-based approach that makes use of local pivot while generating partitions. Single pivot and a branching factor l is used in this approach. Objects in the node are divided into l groups depending on their distance to the vantage point. Along with the vantage point itself, l-1 distance ranges for each subtrees are stored as information. It is possible to divide the space into many partitions by a single distance computation. At query time only one distance calculation is performed per node. However when the dimensionality increases, vp-Tree loses its effectiveness since objects tend to cluster around a single distance value. Therefore many objects become at the same distance to the vantage point, and the distance to the vantage points loses its importance.

The mvp-Tree improves vp-Tree by using two vantage points per node. The partitioning process continues with the second vantage point after the first partitioning. In second partitioning the same branching factor l is used. Therefore there are l^2 subsets. Different distance ranges are used for each partition obtained from the first partition. In this way, each subset has nearly the same number of points, which maintains the balance of the tree. This causes more space consumption per node.

2.1.2 Pivot Based Methods

In pivot based methods, pre-computed distances between a subset of objects called *pivots*, and the rest of the objects are stored in distance matrices. At the query time, this distance information is used in the elimination of the candidate objects. Index structure stores $k \times n$ distance values, where k is the number of pivots, and n is the number of objects in the dataset. As the number of the pivots used increases, the cost of the construction time and storage requirements also increase. However query performance can be improved in terms of number of distance computations by using more pivots at the construction time.

One of the earliest methods, AESA, uses all objects as pivots. The distances of n database object to each other is stored in an $n \times n$ matrix. At the construction time, n *(n -1)/2 distances are computed. At query time, this information is used to perform the elimination. For large datasets, this method is not effective because of high space requirements and construction cost.

LAESA solves the problem of high space requirements of AESA by using only a subset of objects selected as pivots rather than using all of them. The size of the distance matrix is reduced to $n \times m$ where $m<n$. In addition to the improvements over AESA, LAESA keeps the distances to the pivots sorted and performs binary searches to find the objects to be eliminated.

Spaghettis is designed to further decrease the computational overhead. This approach keeps distances of objects sorted for each pivot, moreover a pointer to the same object's distance in the next distance array that is used for distances to another pivot. These pointers are used in tracing the path between arrays to perform the elimination.

FQA, one of the recent pivot-based methods, reduces computational overhead and it does not require additional storage by storing less precise distance values. However when the dimensions increase the accuracy of pivots decreases.

The pivot based methods achieve better results by requiring higher space and time requirements. Kvp structure solves this problem by keeping only the distances to the promising pivots in the construction phase. While achieving as good query results as other pivot based methods, it reduces space and CPU overhead. Kvp shows that pivots that are closer to or distant from a database object are more effective in terms of elimination. Therefore it gives importance to the selection of pivots, by selecting pivots that are maximally separated from each other.

2.2 Approximate Similarity Search

Efficiency problem of similarity search techniques can be overcome by focusing on the quality-time tradeoff. There are some reasons, which motivate approximate similarity searching as indicated in [25]. The user may not be satisfied with the actual result of the similarity search that is implemented with a distance function. There might be some difference between the similarity the user expects and the distance function used underneath. Users might not agree with the exact results of the query and count some of them as incorrect. Therefore results achieved in less time with some incorrect results might be more preferable. In addition, the user may want to give feedback during query time. Depending on the results of the previous searches, the user may want to redefine queries. The most important of all, even if the user is satisfied with the results of the query, it may be preferable to get faster but approximate result.

We can categorize the existing approaches for approximate similarity search into three groups as in [25]: *Approaches that reduce the size of data objects*, *approaches that reduce the size of data set*, and *approaches which guarantee on the result query.*

2.2.1 Approaches that reduce the size of data objects

These types of approaches generally use the techniques of dimensionality reduction based on idea that the most important information can be represented with a few dimensions. Linear Algebraic Methods such as Discrete Fourier Transform can be used in dimensionality reduction. Another common approach is *VA-file* [31], which contains approximation of vectors based on a fixed number of bits. *FASTMAP* [19] is another important method in this category. The idea is to map a set of objects from a generic metric space to a Euclidean space with a user defined dimension value. A distance matrix that holds the *EuclideanDistance* between the objects in the vector space is used to project the objects in a vector space. Quality of the performance of the approximation depends on the number of dimensions of target vector and distance matrix.

2.2.2 Approaches that reduce the size of data set

Approaches in this group can further be classified into two categories according to the strategies they use while reducing the size of data set: *Early Stopping Strategies* and *Aggressive Pruning Strategies*.

In Early Stopping Strategies, the algorithm stops according to a stop condition such as the maximum cost to be paid or a distance value to be reached. Although the correctness of the algorithm can be improved, after some iteration steps, the improvements on the correctness are negligible in comparison to the cost of the query. Practically algorithm generally stops when the chance of obtaining better results decreases. Stop condition is the factor that affects the quality of the query.

Approaches in Aggressive Pruning Strategies, use probabilistic bounds to eliminate regions of metric space, which are unlikely to contain results. *BBD-Tree* [2] index structure belongs to this category. It is main memory index that responses the k-NN queries in a poly-logarithmic time with the number of objects in the dataset. In this structure regions are represented with nodes of the tree, where each node has pointers to the other nodes. This method follows an aggressive pruning strategy via reducing query radius by a factor with respect to the radius used for exact search, in order to prune tree nodes.

Amato et al. proposed a proximity based approximation that uses proximity measures in pruning areas [1]. The largeness of the overlapped area between two ball regions does not always mean that a large amount of data exists in the intersection of these regions, since this amount depends on data distribution. There may be large amounts of data in a small intersection area and a small amount of data in a large intersection area. The proximity measure is used in the decision of elimination of tree nodes even if their bounding regions intersect with the query region. Probabilistic approach is used when analyzing the proximity of two ball regions. The aim is to discard data regions with small probability of sharing objects with the query region. Proximity measurement is an important factor to get accurate results for the approximate similarity search. Since regions that contain qualifying objects may be discarded, it as an approximate approach. This approximation solution can be used for both range and nearest neighbor queries.

P-Sphere tree [20] is another example. It is a 2-level index structure for nearest neighbor approximate search. The lead node closest to the query point is accessed when finding the nearest neighbor of query. Simple linear scan of objects contained in such node is performed to solve the query.

2.2.3 Approaches which guarantee on the result query

Another assessment criterion of approximation approaches is the guarantee of quality that the algorithms have. Some algorithms use only heuristic conditions in approximation without defining a formal bound on the error. FASTMAP can be given as an example to this category, since no guarantee is given on the error. Some algorithms have an upper bound on the error. BBD-Tree gives a deterministic guarantee since the error cannot exceed ϵ, which is used for reducing the query radius. Some algorithms gives probabilistic guarantee by using distribution of data to calculate the error bound. DBIN [4], which is used for k-NN queries, is an example of this category. DBIN is a 2-level index structure. Dataset is divided into clusters, where the objects can be modeled by a Gaussian distribution. At the query time, the cluster that best fits the query object is searched. If the probability that k-NN have not been found yet is higher than a threshold, the search continues for the remaining clusters.

Chapter 3

Methods

This chapter includes various methods and techniques used in different approximation algorithms, which are explained in Chapter 4. Since many of the algorithms make use of some common concepts and methodologies, we introduce them before the algorithms themselves.

3.1 Index Structure

In applying the algorithms developed, the Kvp Structure [8] is used with a minor variation. The Kvp Structure is based on the following idea: The effectiveness of a pivot in eliminating a database object is related to the distance between the pivot and the database object. The pivots those are closer or farther to a database object are proven to be more effective in the elimination of that particular database object in [7]. Hence, in order to benefit from the memory and performance improvements, the Kvp Structure is used in the implementation of the algorithms. For instance, among 100 pivots used for Corel database we made use of 10 most promising pivots in some of our experiments. This means for every database object only a small portion of the pivot distances are stored, which results in less memory usage and better computational performance.

One slight difference of the structure we use with the Kvp is in the determination of pivots. Kvp applies a reasonable methodology in deciding for the pivots to use for a particular database. In order to determine as effective as possible pivots for all the objects in the database, the pivot selection schema chooses the object that is farthest from all the pivots selected so far. Instead of making use of this pivot selection mechanism, all the pivots are selected randomly in this thesis for the sake of implementation simplicity. We initially decided to store approximately 0.2% of total number of dataset objects for all datasets as pivots. After performing initial experiments, we further decreased the number of pivots to 0.02% in order to decrease the effect of the index structure in the elimination of database objects as explained later in Section 5.1.

3.2 Global Distance Distribution

Distance distribution of a dataset is one of the tools that we benefitted in the approximation of range query results. Use of distance distribution has attracted the interest of other researches such as [11], [14], [27], [33]. In [14], the use of distance distribution in metric spaces is claimed to be the counterpart of data distribution in vector spaces. In this study, the view of the whole distance distribution for a particular database object along with the discrepancy of distance distributions for different objects are emphasized. The study in [33] uses the findings of the previous study and defines the concept of "representative distance distribution" for using in approximate queries in metric spaces. In [27], it is mentioned that the distance distribution of the items in the dataset is expected to be very close to the distance distribution of the items in the query set. Hence, the distance distribution of a query object can be approximated by the distance distribution derived from the training set. In this thesis, we used a similar approach to that of the last study mentioned instead of dealing with the relativity of distance distribution from the view point of individual items.

In our study, the distance distribution of each data set used in the experiments is constructed by using the distances between all the database objects and the pivots, already computed for the construction of the index structure explained in 3.1. The maximum distance computed is divided into a pre-defined number of intervals. A histogram is constructed to hold the number of distances falling into each such interval. An illustration of the construction of such a distance histogram is shown in Figure 3.1 below.

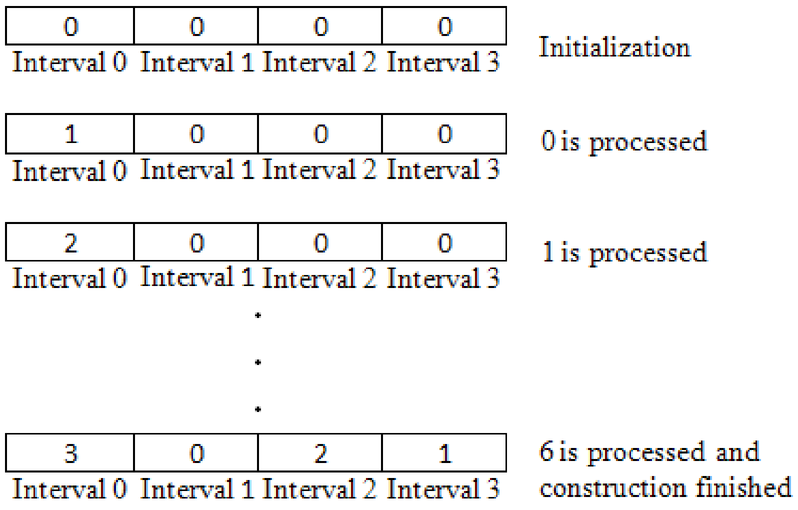

0 0 0 0
Interval 0 Interval 1 Interval 2 Interval 3
Initialization
1 0 0 0
Interval 0 Interval 1 Interval 2 Interval 3
0 is processed
2 0 0 0
Interval 0 Interval 1 Interval 2 Interval 3
1 is processed
3 0 2 1
Interval 0 Interval 1 Interval 2 Interval 3
6 is processed and construction finished

Figure 3.1: Distance distribution histogram construction for distances={0,1,1,4,5,6} and interval length=2, where min. and max. distances are 0 and 6.

We performed experiments with 3 different data sets. In the figure below, distance distribution histograms for these datasets are shown as line graphs. Distance distribution of each data set is divided into 200 intervals, where interval lengths vary. Corel dataset has 0.01, Nasa has 0.014, and Gaussian data set has 0.45 interval length. Corel dataset shown on the left has negative skew, where as the right most data set, Gaussian, has a slightly positive skew. The data set shown in the middle, Nasa data set, has no skew, thus holding the characteristics of a symmetric distribution.

Some of the algorithms to be mentioned do not make use of the distance distribution array directly; but rely on cumulative distance probability of a distance provided. As a result, an array holding the cumulative distance probabilities for each interval is constructed from the distance distribution histogram as follows:

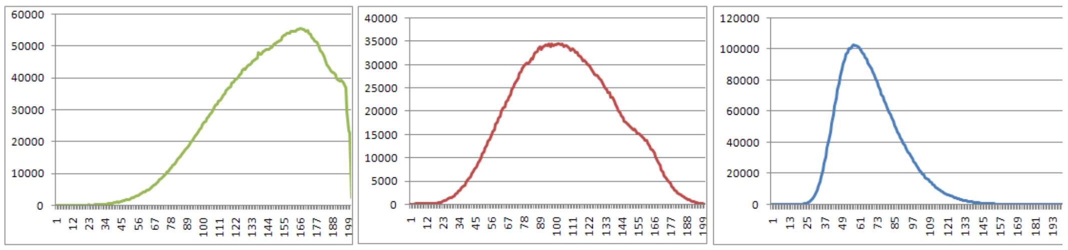


Figure 3.2: Distance distribution for Corel, Nasa, and Gaussian data sets in order. The x-axis is the interval number, where the y-axis represents the number of distances falling into the corresponding interval.

```
method createCumDistProbArray($distanceDistHist$)
1) $sum$ := 0
2) $cumDistProbArray$ :=new float array
3) for each $i$ smaller than size of $distanceDistHist$ do
4)     increment $sum$ by $distanceDistHist[i]$
5)     assign $cumDistProbArray[i]$ to $sum$ divided by total # of distances
```

Figure 3.3: The algorithm explaining construction of cumulative distance probability array from distance distribution histogram.

The approximation algorithms make use of this cumulative distance probability array in calculating the cumulative distance probability for a given *distance*, which is denoted by F(*distance*). The algorithm to calculate cumulative distance probability of a given distance is as follows:

3.3 Local Distance Distribution

Although some of our approximation algorithms make use of global distance distribution, some others depend on local distance distributions in the hope of obtaining better approximation. Local distribution is the distribution of distance values for which similar lower and upper bound values are calculated via triangular inequality. In other words, a local distance distribution is a conditional distribution depending on the lower-upper bound values.

The same array structure explained in Section 3.2 for the global distance distribution is reused for the creation of local distance distribution. However, in

```
method calculateCumDistProb($distance$)
1) $result := 0$
2) if $distance$ smaller than 0
3)     return 0
4) $index$ :=biggest integer <= $distance$ divided by $interval\_length$
5) if $index$ >= size of $cumulativeDistProbabilityArray$
6)     return 1
7) $ratio$ := (remainder of $distance$ from $interval\_length$) / $interval\_length$
8) $result$ :=$cumDistProbArray[index] + ratio$ *
   $(cumDistProbArray[index + 1] - cumDistProbArray[index])$
9) return $result$
```

Figure 3.4: The algorithm explaining calculation of cumulative distance probability for a target distance.

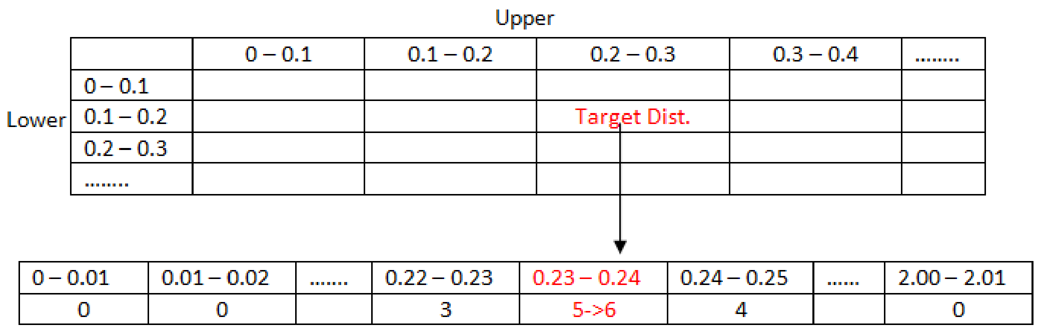
this case there are many distance distribution arrays constructed for lower-upper pairs calculated. The maximum value for the lower bound can be as large as the maximum distance existing in the database; whereas the maximum value for the upper bound can be equal to 2 times maximum distance. The lower-upper values are divided into intervals. We name this interval value as *precision* in order to discriminate it from the interval length of the distance distribution. The greater the *precision* value, the more similar the localized distance distribution arrays are to the global distance distribution array. The smaller it is, more precise the local distance distributions are. However if precision is chosen too small, it will be more difficult to obtain valuable information about the distribution of distances since there will be fewer actual distances per lower-upper pair. For the construction of local distance distribution arrays for each dataset, each pivot is considered as a query object and the distances between this target pivot and all dataset objects are estimated by using the other pivots via triangular inequality. Since, we already have the distance values calculated for the construction of index structure explained in Section 3.1, no new distance calculation except between the pivots is needed. The algorithm below explains the construction of local distribution arrays in this fashion.

The figure below is an illustration of the application of the algorithm lines 9-12 for calculated lower bound value of *0.196* and upper bound value of *0.298* for an actual distance *0.238*. In this particular example precision is 0.1 whereas

```
method constructLocalDistArrays()
1) lowerIntervalCount :=maximum possible lower value / precision
2) upperIntervalCount :=maximum possible upper value / precision
3) localDistArrays :=initialize a double array of distribution arrays of size
lowerIntervalCount × upperIntervalCount
4) for each pivot p in pivots
5)    remainingPivots :=pivots - {p}
6)    for each dataset object o
7)       actualDistance :=distance btw p and o
8)       calculate lowerBound and upperBound for actualDistance
by using remainingPivots via triangular inequality
9)       lowerIndex :=lowerBound / precision
10)      upperIndex :=upperBound / precision
11)      targetInterval :=the interval actualDistance falls in
localDistArrays[lowerIndex][upperIndex] distribution
12)      increment the value stored in targetInterval
```

Figure 3.5: The algorithm explaining construction of local distance distribution arrays for data sets.

interval length of distance distribution is 0.01. The lower value falls into interval 0.1-0.2, while the upper value falls into 0.2-0.3 interval. After the local distance distribution array is determined, the value in the corresponding interval, to which actual distance falls, is incremented. In this scenario, the value 5 in 0.23-0.24 interval is incremented to 6.



	0 – 0.1	0.1 – 0.2	0.2 – 0.3	0.3 – 0.4	……..
0 – 0.1					
0.1 – 0.2			Target Dist.		
0.2 – 0.3					
……..					

0 – 0.01	0.01 – 0.02	…….	0.22 – 0.23	0.23 – 0.24	0.24 – 0.25	……	2.00 – 2.01
0	0		3	5->6	4		0

Figure 3.6: Illustration of modification of local distance distribution array for lower=0.196, upper=0.298, and actual distance=0.238 values.

3.4 Regression

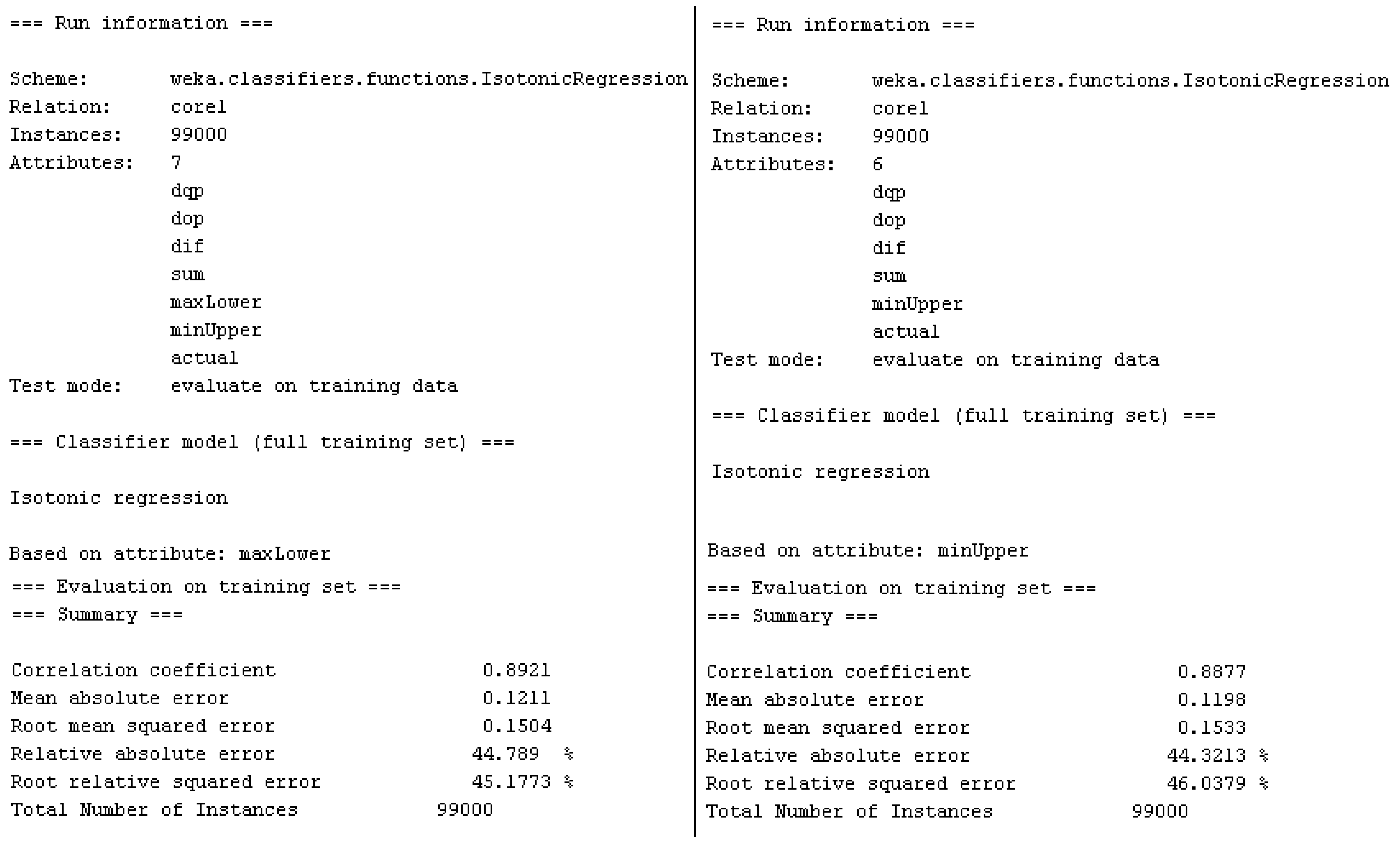
One of the approximation algorithms we propose is based on regression technique that is the estimation of the actual distance with respect to the lower and upper bound values calculated. In other words, a model to estimate required information is needed to be built. This model should depend on the prior knowledge extracted from the data set (used as training data) before any query object (test data) is processed. Similarity search is built upon maximum lower and minimum upper bound values calculated by application of pivoting for all available pivots in the index structure. Hence, the most promising attributes to estimate actual distance between the query object and the database object should be these two. Nevertheless, we applied some experiments on Corel, Nasa, and Gaussian datasets to prove that maximum lower bound value and minimum upper value pair can be used without loss of significant representation power in the estimation of actual distance value.

A similar approach to that applied in the construction of local distributions is conducted in order to create an estimation model from the training data. A random pivot is chosen as the target pivot and treated like a query (test) object and its distance between a randomly chosen training object is tried to be estimated with respect to following attributes: *dqp* is distance between the target pivot and other pivot used in application of triangular inequality. *dop* is distance between the training object and other pivot. *dif* is absolute value of the difference between *dqp* and *dop*; whereas *sum* is the summation of two as the name applies. *maxLower* is the maximum of lower bounds calculated and *minUpper* is the minimum of upper bounds calculated by using all other pivots. For the sake of performance, only a small percent of the pivot-training object distance values are practiced for the datasets.

We performed experiments with Weka [32], an open source machine learning software. The training set used in building the model is used as the training set to evaluate the model in terms of error. Among suitable regression functions available in Weka, *Isotonic Regression* is used initially.

3.4.1 Isotonic Regression

Isotonic regression model picks the attribute that results in least squared error in the estimation of the target attribute. For all of the datasets *maxLower* attribute is chosen as the attribute resulting in least squared error among 6 attributes. We applied Isotonic Regression once more by using the remaining attributes. This time *minUpper* attribute is chosen for all data sets. This is an indication of our claim about using the maximum lower bound and minimum upper bound values in the estimation of the actual distance. The snapshot below is a summary of the results gathered after running of Isotonic Regression for Corel Dataset.



```
=== Run information ===

Scheme:       weka.classifiers.functions.IsotonicRegression
Relation:     corel
Instances:    99000
Attributes:   7
              dqp
              dop
              dif
              sum
              maxLower
              minUpper
              actual
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Isotonic regression

Based on attribute: maxLower
=== Evaluation on training set ===
=== Summary ===

Correlation coefficient                  0.8921
Mean absolute error                      0.1211
Root mean squared error                  0.1504
Relative absolute error                 44.789  %
Root relative squared error             45.1773 %
Total Number of Instances            99000
```

```
=== Run information ===

Scheme:       weka.classifiers.functions.IsotonicRegression
Relation:     corel
Instances:    99000
Attributes:   6
              dqp
              dop
              dif
              sum
              minUpper
              actual
Test mode:    evaluate on training data

=== Classifier model (full training set) ===

Isotonic regression


Based on attribute: minUpper
=== Evaluation on training set ===
=== Summary ===

Correlation coefficient                  0.8877
Mean absolute error                      0.1198
Root mean squared error                  0.1533
Relative absolute error                 44.3213 %
Root relative squared error             46.0379 %
Total Number of Instances            99000
```

Figure 3.7: Summary of results of Isotonic Regression model for Corel Dataset gathered from Weka. The left part is the running performed with all attributes, which resulted in the selection of *maxLower* attribute. The right part illustrates re-running of the regression with remaining attributes, which resulted in the selection of *minUpper* attribute.

3.4.2 Simple Regression Methods

After being sure about *maxLower* and *minUpper* attributes being the best attributes to use in the estimation of the actual distance, we applied various regression algorithms existing in Weka with different sub-sets of 6 attributes. The subsets are formed as follows: All of the attributes are used, only *maxLower* and *minUpper* attributes are used, and lastly remaining 4 attributes are used. The results are compared with respect to root mean squared error and mean absolute error.

Linear Regression, Least Median Squared Regression, and Pace Regression are simple regression methods applied to the datasets. Table 3.1 displays the results gathered for the first three classifiers mentioned. The last two columns of the table are dedicated to "Mean Absolute Error" and "Root Mean Squared Error". Mean Absolute Error is the average magnitude of the difference between the actual and estimated distances. Root Mean Squared Error is the square root of average of the errors squared. It gives large errors more weight than the smaller ones, whereas mean absolute treats each error equally. Despite the difference mentioned, a good regression function should result in small values for both of them.

There are two inferences those should be derived from Table 3.1. The first one is that the attributes *maxLower* and *minUpper* are good enough to use in the estimation of the actual distance, when used together. The derivation of this finding is as follows: Note that all of the regression models resulted in least error when all the attributes are used for each data set. However, it is crucial to notice that the coefficients of *maxLower* and *minUpper* attributes are much larger than the others' coefficients, which means they play a more important role in the estimation. Take the model conducted for Nasa dataset for "LeastMedSq" method as an example. The coefficient for *dqp*, *dop*, and *dif* attributes are '-0.0002', '0.014'. and '0.0096' in order; whereas those for *maxLower* and *minUpper* are '0.4785' and '0.571' respectively. Moreover, difference in terms of both Mean Absoulte Error and Root Mean Squared Error between the cases where all attributes are used and only *maxLower* and *minUpper* are used is so small that it can be neglected. For instance, consider the “Pace Regression” method applied to Corel dataset. The values for errors are ‘0.0966’ and ‘0.1198’ for the first case; where they are ‘0.0973’ and ‘0.1205’ for the latter. Even the errors are same for two cases for Gaussian dataset. However, when the vice versa, the use of all attributes but *maxLower* and *minUpper*, is applied the errors increase to ‘0.2454’ and ‘0.3041’. Hence, use of *maxLower* and *minUpper* values for the estimation yields satisfactory results.

Another finding from the table is that, the use of different regression methods did not result in big differences both in terms of the model itself and the error values. Consider the results of each three method applied to Gaussian dataset with *maxLower* and *minUpper* attributes used. The results of each method is so close to each other that it does not make much more of a difference to choose among one of them.

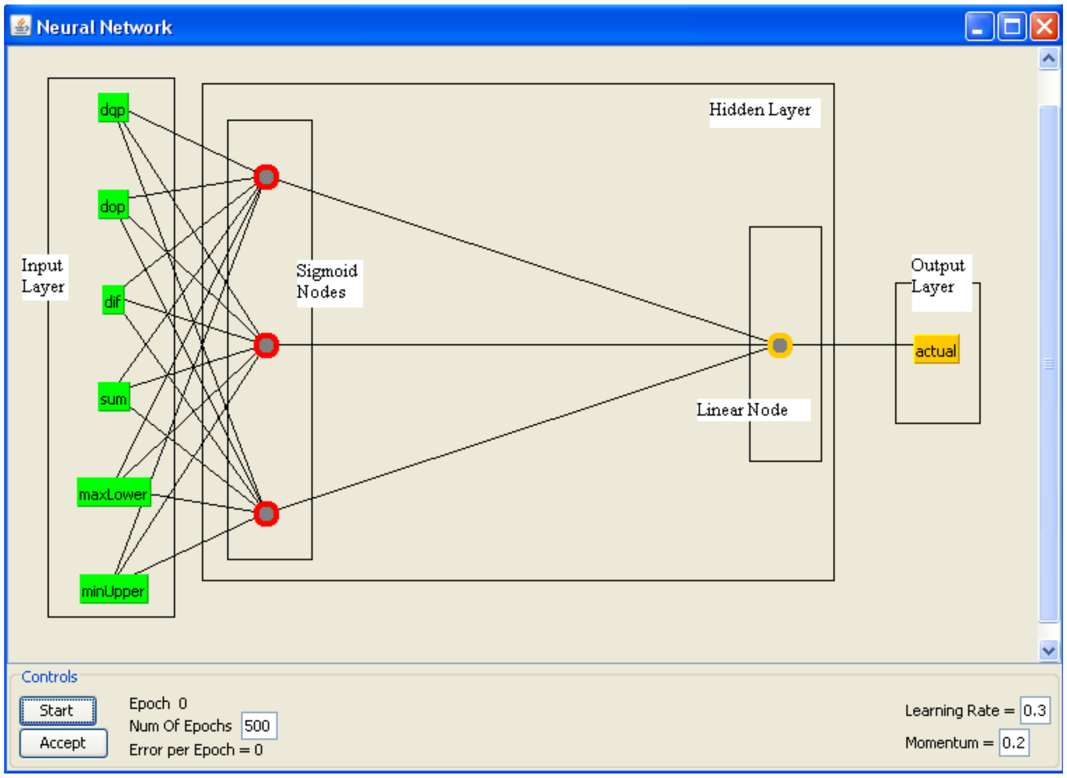
We decided to use a more complicated regression method, Multilayer Perceptron, in order to see if there will be considerable improvement in terms of error or a simpler one is good enough to stick to.

Method	Dataset	Attributes	Model	MAE	RMSE
Linear Reg.	Corel	All	0.03 * dqp - 0.008 * dop + 0.0409 * dif + 0.6124 * maxLower + 0.5381 * minUpper - 0.2361	0.0966	0.1198
Linear Reg.	Corel	maxLower, minUpper	0.6255 * maxLower + 0.5439 * minUpper - 0.2146	0.0973	0.1205
Linear Reg.	Corel	dop, dqp, sum, dif	0.217 * dqp + 0.1311 * dop + 0.4594 * dif + 0.7443	0.2454	0.304
LeastMedSq	Corel	All	0.0336 * dqp - 0.0078 * dop + 0.0392 * dif + 0.6136 * maxLower + 0.5358 * minUpper - 0.2389	0.0966	0.1198
LeastMedSq	Corel	maxLower, minUpper	0.6006 * maxLower + 0.5529 * minUpper - 0.2036	0.0973	0.1206
LeastMedSq	Corel	dop, dqp, sum, dif	0.2351 * dqp + 0.1434 * dop + 0.44 * dif + 0.7196	0.2452	0.3044
Pace Reg.	Corel	All	0.03 * dqp - 0.008 * dop + 0.0409 * dif + 0.6124 * maxLower + 0.5381 * minUpper -0.2361	0.0966	0.1198
Pace Reg.	Corel	maxLower, minUpper	0.6255 * maxLower + 0.5439 * minUpper - 0.2146	0.0973	0.1205
Pace Reg.	Corel	dop, dqp, sum, dif	0.217 * dqp + 0.1312 * dop + 0.4595 * dif + 0.7442	0.2454	0.3041
Linear Reg.	Nasa	All	0.0105 * dop + 0.0084 * dif + 0.4981 * maxLower + 0.5558 * minUpper - 0.1275	0.0697	0.1005
Linear Reg.	Nasa	maxLower, minUpper	0.5049 * maxLower 0.5565 * minUpper - 0.117	0.0702	0.1007
Linear Reg.	Nasa	dop, dqp, sum, dif	0.2071 * dqp + 0.2724 * dop + 0.6614 * dif + 0.4943	0.3041	0.393
LeastMedSq	Nasa	All	-0.0002 * dqp + 0.014 * dop + 0.0096 * dif + 0.4785 * maxLower + 0.571 * minUpper - 0.1306	0.0698	0.101
LeastMedSq	Nasa	maxLower, minUpper	0.516 * maxLower + 0.5385 * minUpper - 0.0979	0.0702	0.1008
LeastMedSq	Nasa	dop, dqp, sum, dif	0.2721 * dqp + 0.3325 * dop + 0.6163 * dif + 0.3588	0.3045	0.3975
Pace Reg.	Nasa	All	0.0108 * dop -0.0012 * dqp + 0.0079 * dif + 0.4983 * maxLower + 0.556 * minUpper - 0.1264	0.0697	0.1005
Pace Reg.	Nasa	maxLower, minUpper	0.5049 * maxLower + 0.5565 * minUpper - 0.117	0.0702	0.1007
Pace Reg.	Nasa	dop, dqp, sum, dif	0.2070 * dqp + 0.2726 * dop + 0.6613 * dif + 0.4943	0.3041	0.3929
Linear Reg.	Gaussian	All	-0.0036 * dop - 0.0022 * dif + 0.5267 * maxLower + 0.5726 * minUpper - 0.3475	0.1987	0.2496
Linear Reg.	Gaussian	maxLower, minUpper	0.5264 * maxLower + 0.5713 * minUpper - 0.3541	0.1987	0.2496
Linear Reg.	Gaussian	dop, dqp, sum, dif	0.0675 * dqp + 0.0266 * dop + 0.5798 * dif + 2.1327	0.5165	0.6564
LeastMedSq	Gaussian	All	-0.0011 * sum - 0.0054 * dif + 0.4931 * maxLower + 0.6039 * minUpper - 0.399	0.198	0.25
LeastMedSq	Gaussian	maxLower, minUpper	0.4793 * maxLower + 0.6111 * minUpper - 0.41	0.1979	0.2502
LeastMedSq	Gaussian	dop, dqp, sum, dif	0.0489 * dqp + 0.0023 * dop + 0.648 * dif + 2.1275	0.5112	0.6623
Pace Reg.	Gaussian	All	0.0034 * dqp - 0.0024 * dif - 0.0033 * sum + 0.5268 * maxLower + 0.5725 * minUpper - 0.3483	0.1987	0.2496
Pace Reg.	Gaussian	maxLower, minUpper	0.5264 * maxLower + 0.5713 * minUpper - 0.3541	0.1987	0.2496
Pace Reg.	Gaussian	dop, dqp, sum, dif	0.0674 * dqp + 0.0267 * dop + 0.5799 * dif + 2.1327	0.5165	0.6564

Table 3.1: Results of Linear Regression, Least Median Squared Regression, and Pace Regression classifiers applied to sampled Corel, Nasa, and Gaussian Dataset distances of size 99000, 50560, and 99000 respectively. Different attribute subsets are compared with respect to the model built, mean absolute error, and root mean squared error.

3.4.3 Multilayer Perceptron

A multilayer perceptron is a kind of neural network, in which units in one layer are connected to the units of the next layer. There are three layers named as input layer, hidden layer, and output layer. In our case input layer consist of the attributes used in the estimation model, and output layer is composed of just the actual distance value estimated. Figure 3.8 displays the multilayer perceptron constructed when all the attributes are used. The red nodes labeled as ‘Sigmoid Nodes’ uses a sigmoidal function for the activation that is transmission of the summation of input values to the next layer if it is greater than a threshold value. The yellow node labeled as ‘Linear Node’ uses a linear function instead of a sigmoidal one.

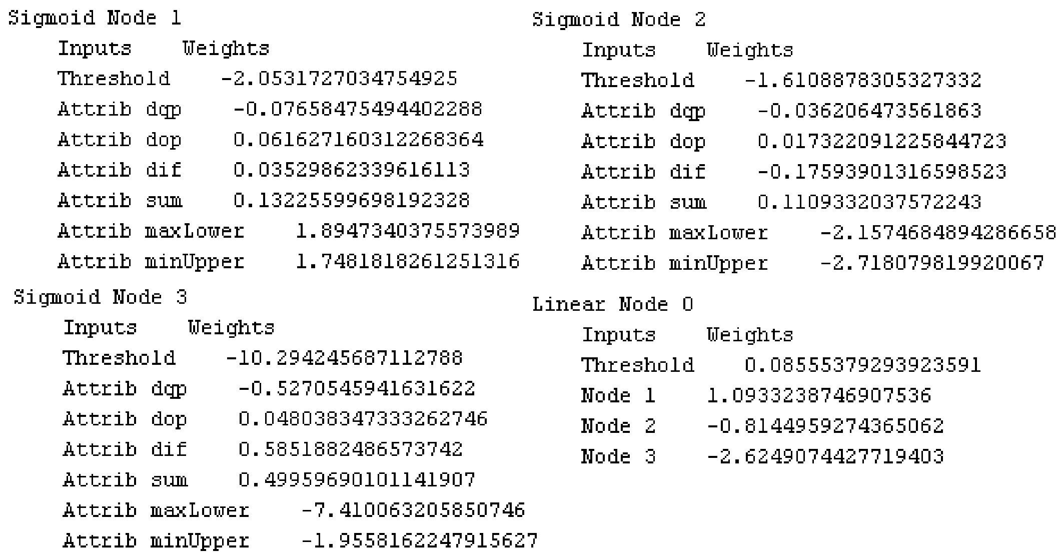


Epoch 0
Num Of Epochs 500
Error per Epoch = 0
Learning Rate = 0.3
Momentum = 0.2

Figure 3.8: Visual representation of multilayer perceptron(Screenshot from Weka).

In a multilayer perceptron, the weights of the connection between units in one layer and the next one are calculated to give as small errors as possible. Figure

3.9 is a summary of the model constructed by Weka for Nasa dataset by using the network visualized in Figure 3.8. Note that, the summary contains weight values for all the connections existing in the model along with the activation values for the nodes of the hidden layer.



```
Sigmoid Node 1
    Inputs    Weights
    Threshold    -2.0531727034754925
    Attrib dqp    -0.07658475494402288
    Attrib dop    0.061627160312268364
    Attrib dif    0.03529862339616113
    Attrib sum    0.13225599698192328
    Attrib maxLower    1.8947340375573989
    Attrib minUpper    1.7481818261251316

Sigmoid Node 2
    Inputs    Weights
    Threshold    -1.6108878305327332
    Attrib dqp    -0.036206473561863
    Attrib dop    0.017322091225844723
    Attrib dif    -0.17593901316598523
    Attrib sum    0.1109332037572243
    Attrib maxLower    -2.1574684894286658
    Attrib minUpper    -2.718079819920067

Sigmoid Node 3
    Inputs    Weights
    Threshold    -10.294245687112788
    Attrib dqp    -0.5270545941631622
    Attrib dop    0.048038347333262746
    Attrib dif    0.5851882486573742
    Attrib sum    0.49959690101141907
    Attrib maxLower    -7.410063205850746
    Attrib minUpper    -1.9558162247915627

Linear Node 0
    Inputs    Weights
    Threshold    0.08555379293923591
    Node 1    1.0933238746907536
    Node 2    -0.8144959274365062
    Node 3    -2.6249074427719403
```

Figure 3.9: Summary of the model constructed by using the network visualized in Figure 3.8.

The multilayer perceptron is applied to all three datasets just like the simple regression methods explained in Section 3.4.2. First with all the attributes, then with *maxLower* and *minUpper* attributes, and lastly with the remaining ones. The results are shown in Table 3.2. The findings of the previous section also apply to multilayer perceptron. In other words, we can conclude that using just *maxLower* and *minUpper* attributes in the estimation model is acceptable in terms of error since there is not a major difference with the case where all attributes are used.

The most important finding is that multilayer perceptron did not give better results than the other regression methods. This is most probably due to the fact that, the model can simply be built with just two attributes where each of them contributes to the model equally roughly. In summary, we decided to use models built by Linear Regression method along with *maxLower* and *minUpper* attributes for our approximation algorithms relying on regression technique.

Dataset	**Attributes**	**MAE**	**RMSE**
Corel	All	0.1019	0.1277
Corel	maxLower, minUpper	0.1213	0.153
Corel	dop, dqp, sum, dif	0.2457	0.3081
Nasa	All	0.0701	0.1005
Nasa	maxLower, minUpper	0.0767	0.1081
Nasa	dop, dqp, sum, dif	0.2991	0.2881
Gaussian	All	0.21	0.2623
Gaussian	maxLower, minUpper	0.2287	0.2868
Gaussian	dop, dqp, sum, dif	0.6994	0.9014

Table 3.2: Results of Multilayer Perceptron applied to Corel, Nasa, and Gaussian datasets in terms of Mean Absolute Error and Root Mean Squared Error.

Chapter 4

Algorithms

This chapter describes various approximation algorithms we propose for the improvement of similarity range search query performance whilst introducing some amount of error in the results. The results of the experiments performed are to be provided and discussed in Chapter 5. Before introducing the algorithms themselves, a classification of possible approximation strategies is provided. The selection of appropriate strategy for each algorithm is mentioned in the corresponding section.

4.1 Approximation Strategies

An approximation algorithm can make approximate decisions in three different ways: Negative, Positive, and Mixed.

4.1.1 Negative Strategy

Negative strategy can be applicable for cases, where it is not tolerated to result in false positive decisions, but false negatives can be tolerated up to a limit. In other words, in this type of the algorithm either "out" or "not decided" decisions are made. By using this strategy, one can be sure to achieve 100% precision, whereas the recall value will vary depending on the elimination rate of the algorithm. All the database objects returned to the user as the query result will be actually inside the query range, since they will be included in the query result either by the index structure decision or distance calculation. However, some of the database objects will be mistakenly removed from the result set due to false out decisions made.

4.1.2 Positive Strategy

Positive strategy can be applicable for cases, where false positives can be tolerated, but false negatives must be strictly forbidden. As opposed to the negative strategy, the users will achieve varying precision values; whereas the recall value is guaranteed to be 100%. The reason is that, positive strategy does not discard any correct results; but introduces erroneous objects to the result set returned to the user. In other words, only “in” or “not decided” decisions are made.

4.1.3 Mixed Strategy

Mixed strategy is a combination of positive and negative strategies. In other words, both “in” and “out” decisions should be made by the algorithm, which results in varying values for both of precision and recall.

4.2 Radius Shrinking Based On Distance Distribution (RSDD)

Shrinking the radius is an effective technique to eliminate the database objects for which a decision could not be made by the application of triangular inequality. Let the radius be denoted by r and the reduced radius by r’. The database objects for which the lower bound calculated is smaller than r can now be eliminated if that lower bound is greater than *r'*. The figure shown below displays database objects whose lower bounds are smaller than r but greater than *r'*. By reducing the query radius, the approximation algorithm will now decide for all of these objects as "out". This means the algorithm should result in correct decisions for the database objects shown with green dots; but should make erroneous decisions for the ones shown with red dots.

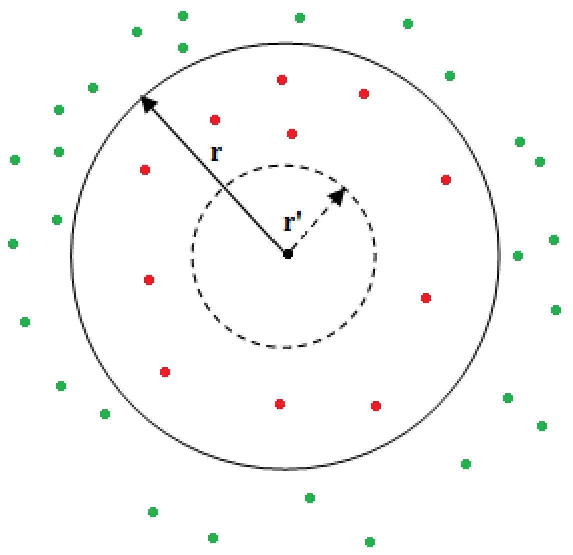


Figure 4.1: Shrinking the radius should result in the elimination of objects shown with green dots successfully, but the objects shown with red dots are also eliminated mistakenly.

Radius shrinking is a technique that was previously applied in different researches, i.e. [2], [11], [13], [15], and [33]. In these researches, either the radius is shrunk by multiplying it with Ω, where Ω is between 0 and 1 or by dividing it to $(1+\epsilon)$, where ϵ is a user supplied error rate.

In our case, we applied radius shrinking with respect to the distance distribution explained in Section 3.2. A user supplied parameter, α, is taken and the inverse of the distance distribution is used to obtain *r'* that is smaller than r. In other words, *F(r')*/*F(r)*=α. Hence *r'* is equal to $F^{-1}(F(r) \times \alpha)$. The application of this approach is visualized in Figure 4.2 given below for Corel dataset, where r is equal to 1.4 and α is equal to 0.5. Radius is reduced to 1.15, which means the database objects whose lower bounds are between 1.4 and 1.15 are decided to be outside 1.4 range of the query object. A formal summary of the algorithm is provided in Figure 4.3. As explained above and seen in the algorithm, RSDD is inherently a negative strategy algorithm.

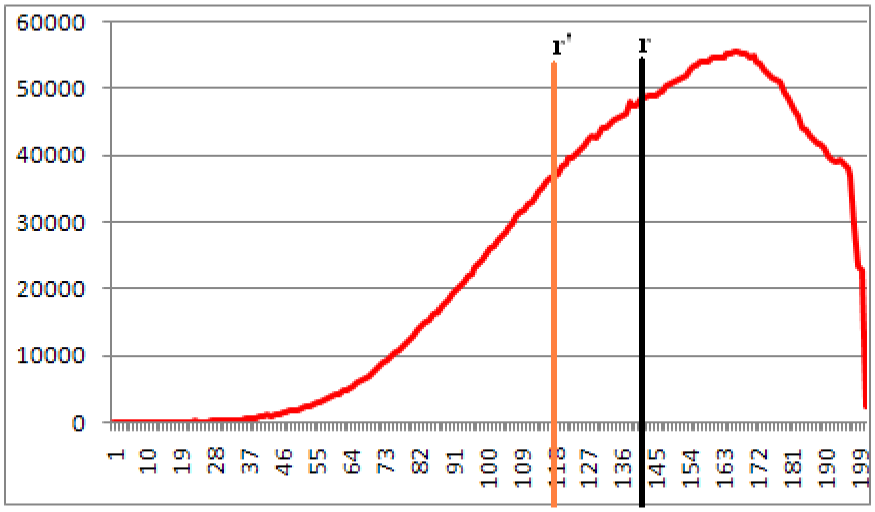


Figure 4.2: Radius is reduced from 1.4 to 1.15 with respect to the distance distribution of Corel Dataset when α is equal to 0.5.

```
method APPLYRSDDALGORITHM($q, o, r\alpha$)
1) $lowerBound$ :=maximum of lower bound values calculated for each pivot
2) $upperBound$ :=minimum of upper bound values calculated for each pivot
3) if $lowerBound > r$
4)    decide for OUT and return
5) if $upperBound <= r$
6)    decide for IN and return
7) $r' := F^{-1}(F(r) \times \alpha)$
8) if $r' > lowerBound$
9)    a decision cannot be made and distance calculation is needed
10) else
11)    decide for OUT and return
```

Figure 4.3: The pseudocode explaining the application of RSDD approximation algorithm for query object q, database object o, radius r, and user-supplied α.

4.3 Conditional Probability Based Elimination (CPBE)

This section is dedicated to "Conditional Probability Based Elimination", which is the first of the approximation algorithms performing elimination in regards to particular query-database object pair. The term conditional comes from the fact that the elimination of database objects for which index structure did not help is performed in regard to a user supplied threshold value, Ω. The probability calculation is performed by using either the global or local distance distributions explained in Sections 3.2 and 3.3 respectively.

The algorithm starts with the application of triangular inequality in order to decide for a database object o to be inside or outside of radius r of a query object q like all other algorithms. Let the lower bound calculated for the distance doq by using all pivots available be l and upper bound be u. If l is smaller than r and u is greater than r then there comes the application of CPBE for making a decision approximately.

The cumulative distance probabilities of r and l are calculated and let them be represented by *F(r)* and *F(l)* respectively. If the difference between *F(r)* and *F(lower)* is smaller than Ω, then the algorithm decides for o to be outside of r distance of q. Otherwise, the algorithm still cannot decide and actual distance between o and q is needed to be calculated. The logic behind this approach is derived from the fact that the probability of o being outside of r radius of q is inversely proportional to the distance between l and r. In other words, as l gets closer to r, the probability of dqo being greater than r increases.

The visualization of the region between l and r is provided in the figure below, which is the distance distribution of Corel dataset for intervals of length 0.01. In the figure radius is equal to 1.4 and lower bound is around 1.05. The probability of a distance being in the shaded area is equal to *F(r) - F(l)*. If this probability is small, CPBE tends to decide for database object o being outside r range of the query object o.

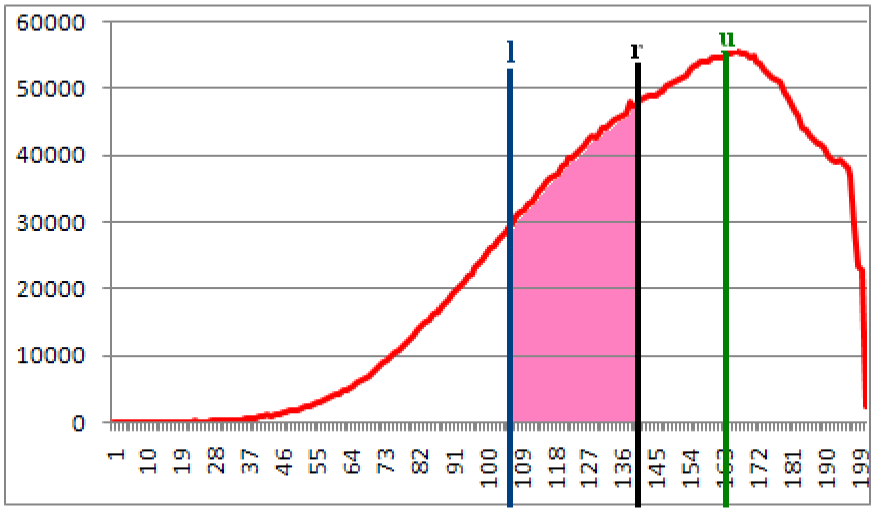


Figure 4.4: Distance distribution for Corel Data Set, where sample r, l, and u values are shown. The shaded area represents the distances falling between r and l.

Among possible strategies explained in Section 4.1, we decided to implement only negative strategy of CPBE algorithm. The reason is as follows: In a range query, the radius value is generally selected to retrieve only a small portion of all the database objects, such as 1%, 3% or 5% at most. As a result, most of the objects is outside the range of the query object. In such a case, a positive strategy should decide on only a very small portion of all the objects, since it cannot make any "out" decisions. So, actual distance calculation is needed for most of the objects not eliminated by the index structure and the cost of the query is not decreased as desired. A mixed strategy might have been preferred, if only the parameter value Ω should have been tuned to give good results for both of positive and negative strategies.

4.3.1 Relative Probability

There should be some improvements related to using *F(r)* - *F(l)* value in CPBE algorithm. For instance consider the following case: If *F(r)* itself is smaller than Ω, then negative strategy decides for "out" regardless of lower value. In order to overcome such a situation, we decided to improve CPBE results by using (*F(r)* -

F(l))/(*F(u)* - *F(l)*) value instead of *F(r)* - *F(l)*.

We call this approach *RelativeProbability*, since the difference between the cumulative distance probabilities of radius and lower bound values is divided to the difference between cumulative distance of upper bound value and that of lower bound value. Using relative probability can be justified by the following argument: As indicated above for Figure 4.4, the probability of a distance being inside the shaded area is equal to the total number of distances falling in that region divided by the total number of distances. However, one little flaw in this calculation is that some of the distances are counted redundantly even though there is no chance for them to occur. The upper value, shown with the green line in the figure, displays the maximum possible value, somewhere around 1.62. This means, the values greater than 1.62 has no chance to occur for this particular case, thus there is no point in including them in the calculation of the probability of the shaded area. Similarly, the lower bound value, shown with blue line in the figure, displays the minimum possible value, somewhere around 1.04. Hence, the distance *doq* can only take values between 1.04(lower bound) and 1.62(upper bound). As a result, the probability of the shaded area can be calculated by total number of distances between lower and radius values divided to total number of distances between lower and upper values.

4.4 Boundary Guided Error Sensitive Elimination (BGESE)

The algorithm that uses regression techniques explained in Section 3.4 is named "Boundary Guided Error Sensitive Elimination". BGESE uses both of the lower and upper bound values in the estimation of the actual distance between a query object and a database object. The "Model" field of the rows of Table 3.1, which contain *maxLower* and *minUpper* as "Attributes", are used as the estimation model by BGESE algorithm for the corresponding dataset. Moreover, "Mean Absolute Error" fields of the same rows are used in making the decision whether or not the distance between a query object and a database object is smaller than the radius value specified. We decided to parameterize the error sensitivity and give user the opportunity to perform aggressive (where the error of the model is ignored) approximation, cautious approximation (the error rate is at least 1), or an approximation in between two extremes. In other words, a parameter β is taken from the user along with the query object itself and the radius. β parameter determines the effect of the mean absolute error in making a decision. The pseudocode given below summarizes the BGESE algorithm.

```
method applyBGESEAlgorithm($q, o, r, \beta$)
1) $lowerBound$ :=maximum of lower bound values calculated for each pivot
2) $upperBound$ :=minimum of upper bound values calculated for each pivot
3) if $lowerBound > r$
4)   decide for OUT and return
5) if $upperBound <= r$
6)   decide for IN and return
7) $estimatedDistance$ :=use $lowerBound$ and $upperBound$ in estimation of $doq$
8) if $|estimatedDistance - r| < \beta \times MAE$
9)   a decision cannot be made and return
10)if $estimatedDistance > r$
11)  decide for OUT and return
12)else
13)  decide for IN and return
```

Figure 4.5: The pseudocode explaining the application of BGESE approximation algorithm for query object q, database object o, radius r, and error sensitivity β. Note that MAE is "Mean Absolute Error" of the specific dataset.

The BGESE algorithm given above illustrates a Mixed Strategy variant; where lines 1-11 makes up Negative Strategy and lines other than 10-11 composes the Positive Strategy. We implemented two variants of BGESE algorithm: Mixed and Negative. We discarded the Positive Strategy variant due to the same reasons explained in Section 4.3 above.

Chapter 5

Results

This chapter starts with an explanation of how the experiments are performed and a definition of the evaluation criteria used in comparing different algorithms. The results obtained for different datasets with different configurations are provided as graphs in order to ease the comparison with respect to the evaluation criteria. Finally, interpretation of the results and an overall discussion are provided.

5.1 Experiments

We performed experiments with three different datasets: Corel (32-featured), Nasa(20-featured), and Gaussian(16-featured). Among these three, Corel and Nasa are real life examples obtained from [17] and [24] respectively. Gaussian dataset is an artificial dataset whose features are obtained from a Gaussian distribution. For Corel dataset we used 49900 database objects and 100 query objects. The corresponding numbers for Nasa and Gaussian are 39270, 80 and 48000, 100 in order. The number of pivots to be used by the underlying pivot-based structure is another important parameter that should have an impact on the performance of the approximation algorithms. Initially we decided to use 100, 80, and 100 pivots for Corel, Nasa, and Gaussian datasets respectively and store 10, 8, and 10 closest pivots per database objects in the same order. However, the initial experiment results revealed that such a configuration might result in a very high elimination rate by the index structure itself, and the effect of approximation algorithms might be relatively small. Hence, we decided to go along with just 10, 8, and 10 pivots. We used Manhattan distance for Corel and Gaussian datasets; whereas Eucledian distance is used for Nasa dataset.

Among the algorithms described in Chapter 4, RSDD decreases the radius value with respect to the distance distribution of the dataset and does not take into account the features of individual query object. Due to this characteristic, we accepted RSDD as the base algorithm whose performance is expected to be outperformed or achieved by the remaining algorithms. The reason is that, all the other algorithms are more sensitive to the query object than RSDD is.

For CPBE algorithm we decided to use three different variations. The first one is named "Global Negative CPBE" (GN-CPBE). It uses the global distance distribution and negative strategy, but does not benefit from the relativity explained in Section 4.3.1. The second variant is called "Global Relative Negative CPBE" (GRN-CPBE). It is very similar to the first one, but relies on relative probability concept. The last variant is "Local Relative Negative CPBE" (LRN-CPBE). It differs from the second one by using local distance distributions instead of a single global distance distribution.

For BGESE algorithm, two different versions are used in experiments. They are named as "Negative BGESE" (N-BGESE) and "Mixed BGESE" (M-BGESE). The first one makes use of negative strategy, while the latter adopts a mixed one. We decided not to apply positive strategy for both of BGESE and CPBE algorithms. The reason is that no major improvement in terms of elimination of the database objects can be achieved with positive strategy alone as explained in Section 4.3.

We executed the variation of the algorithms mentioned above for all datasets for 3 different radius values. The radius values are calculated so as to cover 1%, 3%, and 5% of all the database objects. The effect of the index structure in eliminating the database objects decreases as the covering radius increases. As a result, the scales of the graphs for different radius values differ from each other in order to emphasize the difference of the performance between different algorithms.

5.1.1 Evaluation Criteria

All but one of the algorithms we used in the experiments adopt negative strategy described in Section 4.1.1. This means that they should result in 100% precision no matter how. If all of our algorithms were to use negative strategy, we should have compared them with respect to the elimination rate for the same recall value. Due to the existence of M-BGESE algorithm, which uses a mixed strategy and should result in varying recall and precision values, such a comparison is not possible. As a result, we decided to compare the algorithms with respect to the elimination rate for the same or relatively close F1-score values. F1-score takes into account both of the recall and precision values. We adopted a balanced F1-score that uses the harmonic mean of precision and recall. More formally, we used the following formula in our calculations:

$$F1 - score = \frac{2 \times recall \times precision}{recall + precision}$$

The elimination rate is equal to the ratio of total number of database objects eliminated by the index structure and the approximation algorithm in total to the total number of database objects. It can be formulated with the following formula:

$$EliminationRate = \frac{\#\ of\ db\ objects\ \ - \#\ of\ distance\ computations\ needed}{\#\ of\ db\ objects}$$

5.2 Results and Comparison

This section includes scatter graphs, which illustrate the comparison of the approximation algorithms with respect to “Elimination Rate-F1 Score” pair. One common characteristic of all the scatter graphs to be provided is that the elimination rate of an approximation algorithm decreases as F1-score value increases. This means that as less error is desired in the results of a query, the approximation algorithm will become more cautious in eliminating objects. As a result, more distance computations will be required. The elimination rate will become closer to that of index structure's as F1-score approaches to 1.

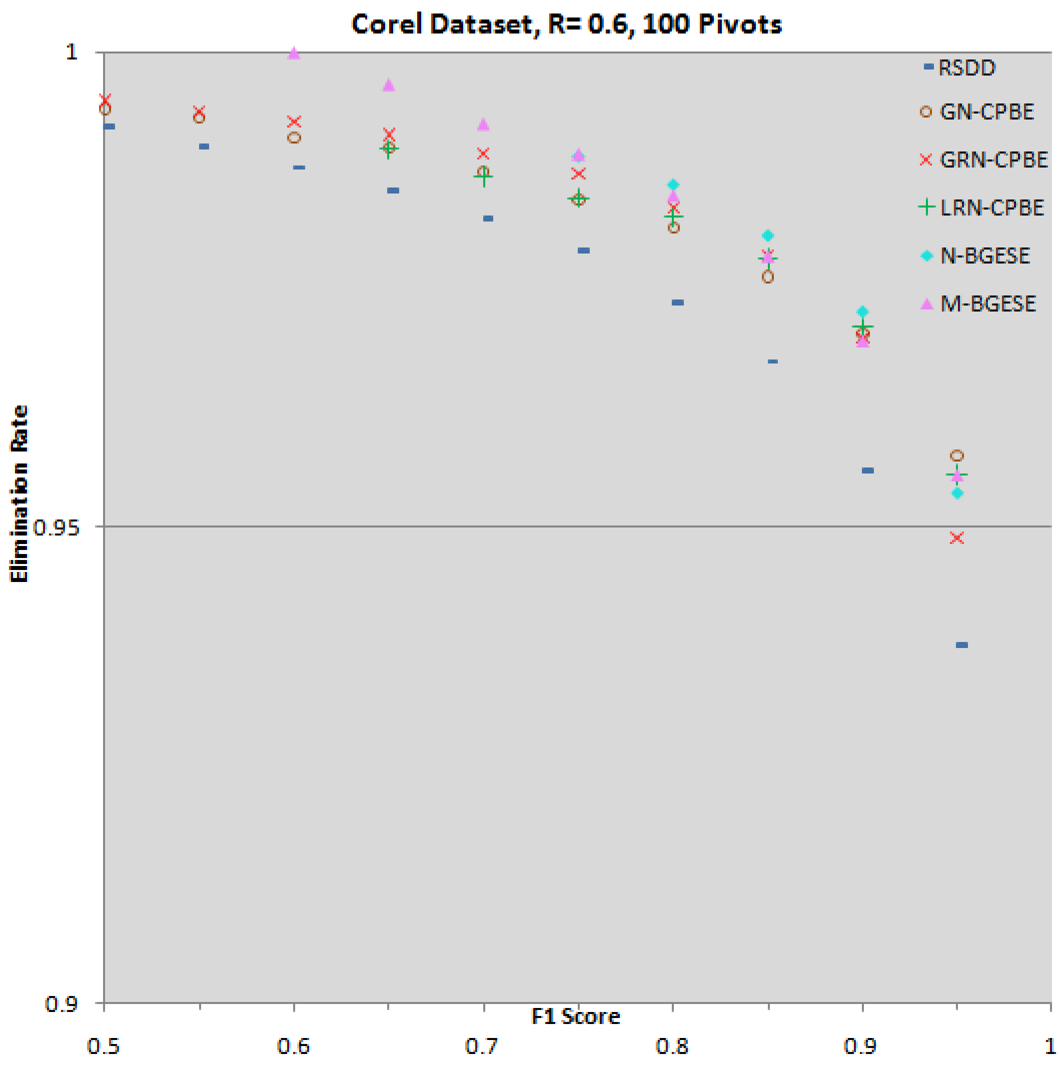


Figure 5.1: Elimination Rate-F1 Score comparison for dataset=Corel, radius coverage=1%, pivot #=100, and Index structure elimination=85.3%

The first of the scatter graphs is depicted in Figure 5.1 above. RSDD algorithm is outperformed by all the other algorithms as expected. M-BGESE algorithm gives the best performance for F1-score values from 60% to 75%. For F1-score values between 80%-90%, N-BGESE seems to outperform the remaining ones slightly. In this particular experiment, the index structure elimination rate is quite high. This means that, the index structure eliminates most of the database objects even without an approximation algorithm's help.

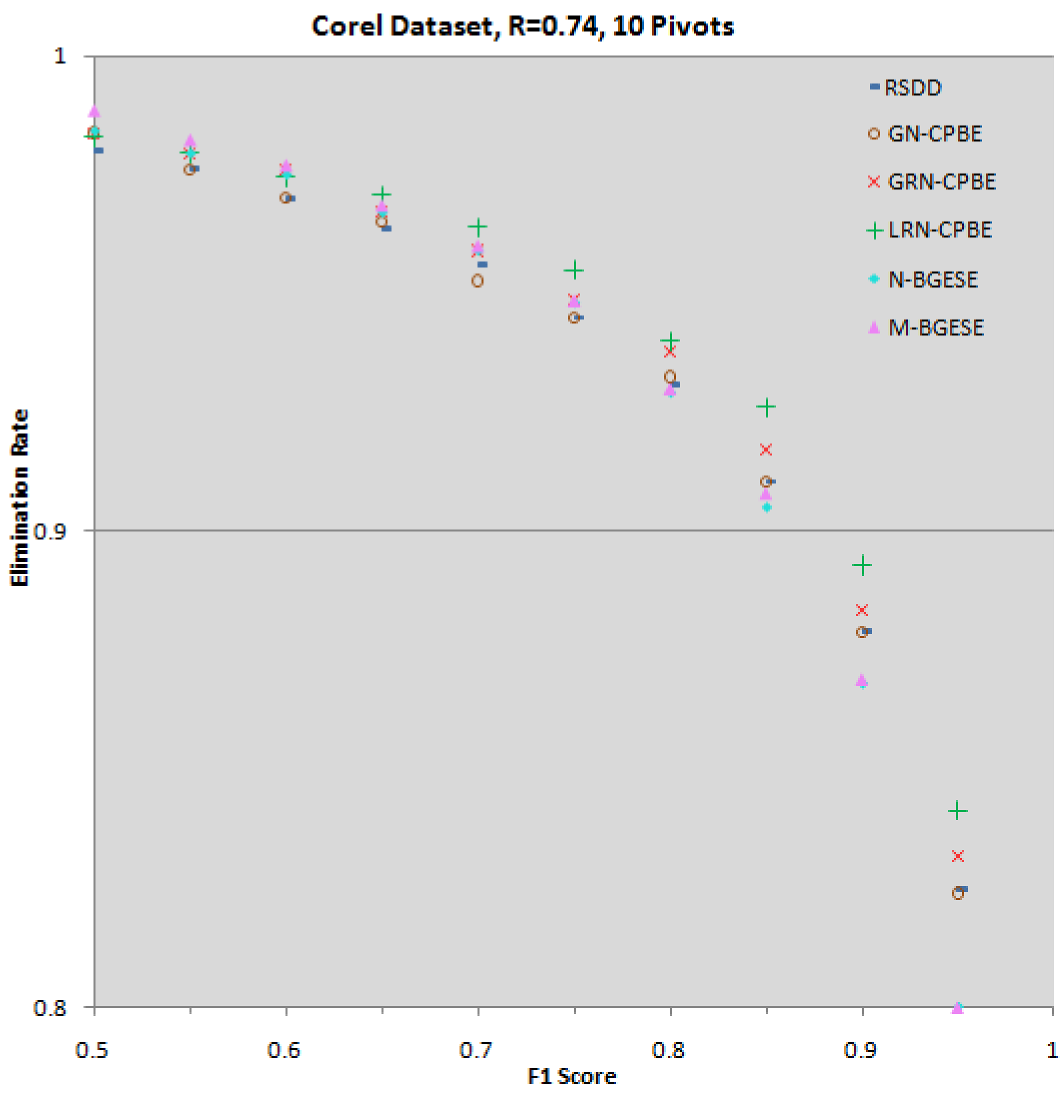


Figure 5.2: Elimination Rate-F1 Score comparison for dataset=Corel, radius coverage=3%, pivot #=10, and Index structure elimination=50%

In order to decrease the effect of the index structure in the elimination, we decided to decrease the number of pivots used and increase the radius. As the number of pivots decreases, the lower and upper bound values calculated between a query object and a database object will become looser. As a result, the index structure will not be able to eliminate as much objects as before.

Figure 5.2 above illustrates the effect of decreasing the pivot number whilst increasing the radius coverage from 1% to 3%. The first observation of the figure is that rates of all the algorithms drop from values above 95% to values around 85%. The main reason is the decrease of the elimination power of the index structure itself. The most outstanding finding about the elimination rate comparison among the algorithms seems to be the decrease of the performance of "BGESE" variants. Both M-BGESE and N-BGESE performs worse than RSDD for F1-score values greater than 80%. This can be due to the fact that the estimation models built for 100-pivot index structure is more representative than the models constructed for 10-pivot structure. In other words, the change of the pivot number of the underlying structure required regeneration of the models listed in Table 3.1. Since the lower bound and upper bound values are looser with 10 pivots, the new estimation models results in greater "Mean Absolute Error" and "Root Mean Squared Error". This means that the estimation models do not model the underlying dataset as well as they did. The most successful algorithm in regard to the evaluation criteria is LRN-CPBE as expected, since it is more sensitive to the dataset and the query object thanks to using local distance distributions. GN-CPBE's performance is very similar to RSDD; whereas GRN-CPBE outperforms RSDD for F1-score values greater than 80%.

Figure 5.3 below displays the effect of increasing the radius coverage to 5%. GN-CPBE's performance is again very similar to RSDD. GRN-CPBE also gives a similar performance to its performance with in Figure 5.2. Until F1-score value of 85% its performance is very close to that of RSDD. After that, its performance does not decrease as much as the other algorithms. It is even the best performer for 95% F1-score value. Even if it is not the best performer for many individual F1-score values, LRN is again the best performer for increased radius value in average. BGESE variants perform better when the radius increases. They always outperform RSDD, where M-BGESE gives the best performance up to 75% F1-score value. The performance of M-BGESE becomes identical to N-BGESE's after F1-score surpasses a certain value. This is due to the reason that the probability of the algorithm to eliminate a database object that is inside the radius of the query object degrades considerably as the algorithm becomes more cautious since only a small percentage of the database objects are inside the query range.

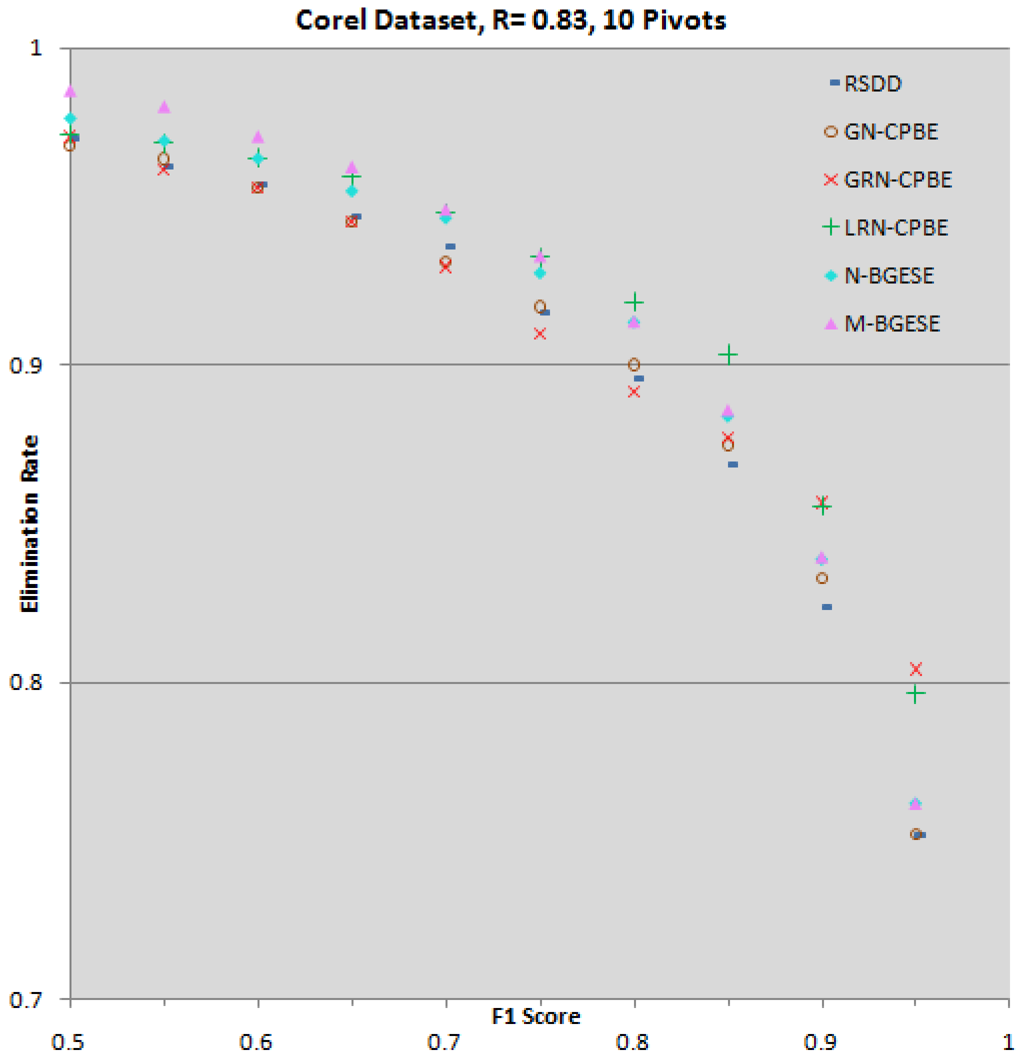


Figure 5.3: Elimination Rate-F1 Score comparison for dataset=Corel, radius coverage=5%, pivot #=10, and Index structure elimination=37.3%

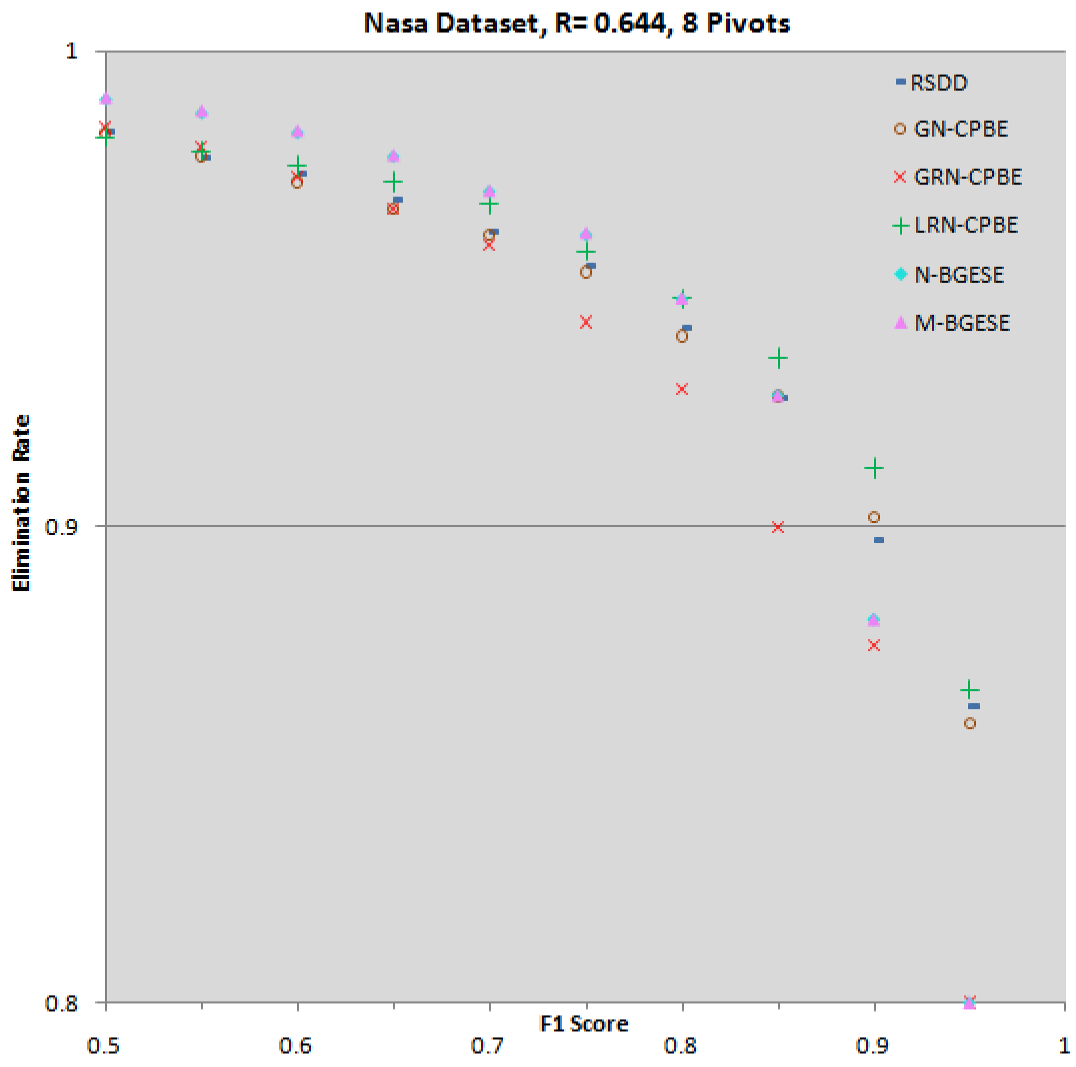


Figure 5.4: Elimination Rate-F1 Score comparison for dataset=Nasa, radius coverage=1%, pivot #=8, and Index structure elimination=47.2%

Figure 5.4 displays the first of the experiment results performed on Nasa dataset. The findings derived from this figure are consistent with the results derived for Corel dataset with minor exceptions. BGESE variants give very identical performance to each other, which means M-BGESE could not succeed to make remarkable number of “IN” type eliminations. BGESE variants give the best performance up until 75% F1-score value and they perform worse than RSDD after 90%. GN-CPBE gives similar performance to RSDD as it was the case for Corel dataset. LRN-CPBE is again the best performer in the average.

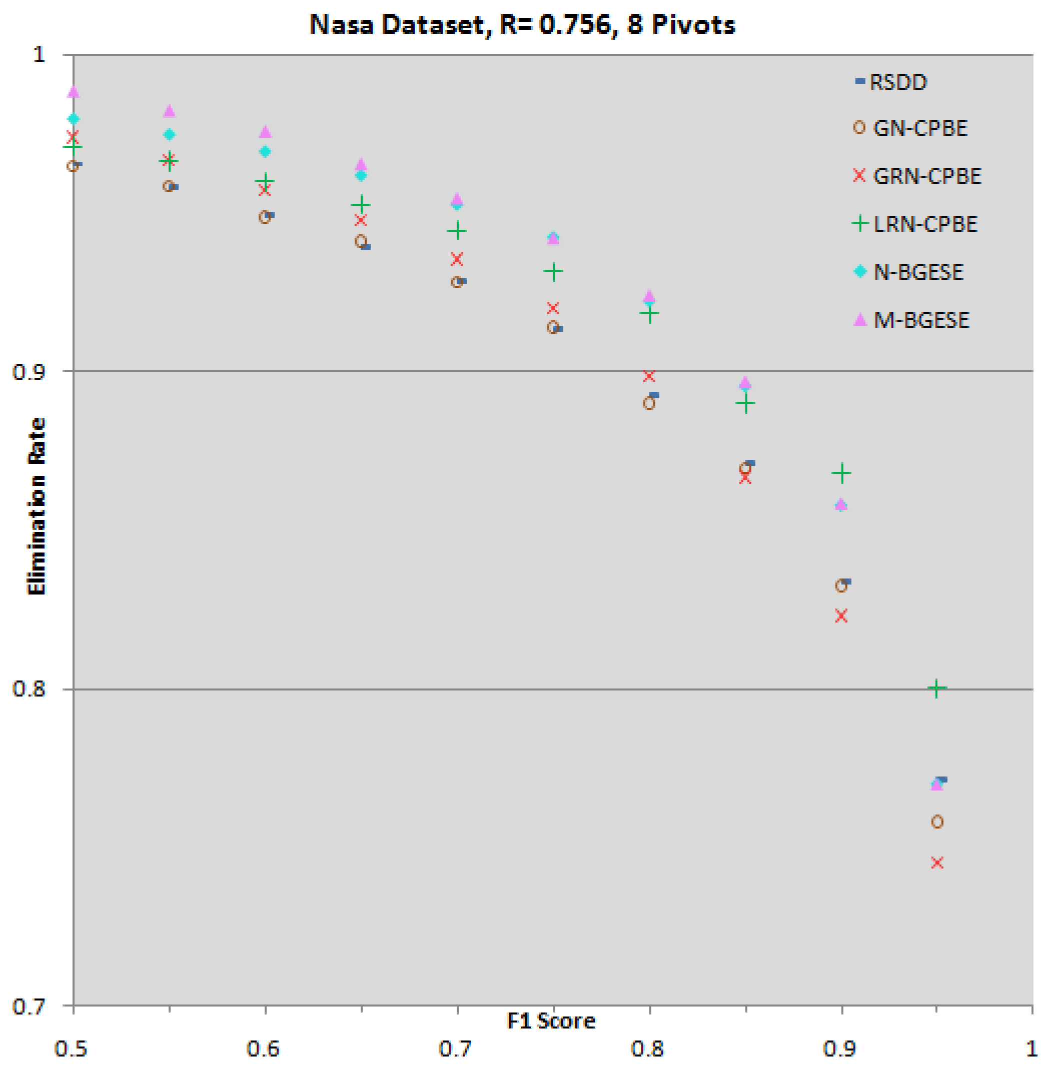


Figure 5.5: Elimination Rate-F1 Score comparison for dataset=Nasa, radius coverage=3%, pivot #=8, and Index structure elimination=32.4%

One of the most important observations is the poor performance of GRN-CPBE. It is surpassed by even RSDD for half of the F1-scores. This is the major difference from the experiments conducted for Corel. We performed more experiments with Nasa dataset via increasing the radius coverage in order to see how the performance of the algorithms change.

Figure 5.5 above includes the result of the experiments performed with increased radius value that covers 3% of the database objects. BGESE variants surpass RSDD almost for all F1-scores, where M-BGESE is the best performer until F1-score of 85%. After that value both variants give identical performances which have been the case for the experiments performed also on Corel dataset. LRN-CPBE is always among the top three performers for all F1-score values and it is the best one after the performance of BGESE variants decrease. GN-CPBE follows the same routine of giving alike performance to RSDD; whereas GRN-CPBE outperforms RSDD until 85% F1-score.

Figure 5.6 below illustrates the change of the performance of the algorithms when the radius coverage increases to 5%. M-BGESE gives the best performance for almost all F1-score values and its performance is outstanding especially for smaller ones. N-BGESE follows M-BGESE for most F1-score values until 95% where it gives an identical performance and outperformed by LRN-CPBE. LRN-CPBE keeps on resulting in consistent elimination rates by beating RSDD for all samples. GN-CPBE keeps on giving alike performance to RSDD; whereas the performance of GRN-CPBE is more determined than it was in surpassing RSDD.

Before continuing discussing the results obtained for Gaussian dataset, it is better to compare the performance of the algorithms for Corel and Nasa datasets. GN-CPBE has been among the most consistent performer for both datasets. It neither achieved an improvement over RSDD nor was surpassed by it. GRN-CPBE has not been a regular performer and it has given inconsistent performances as the dataset and the radius values change. LRN-CPBE has given solid performances and outperformed RSDD no matter what the dataset or the radius value is. BGESE variants have been more successful when the radius increases; but their performances decreased more than the others when F1-score value passes 85%. We see that BGESE variants have been more successful for Nasa than they have been for Corel. This can be due to the reason that the estimation model obtained via regression technique is more successful for Nasa dataset.

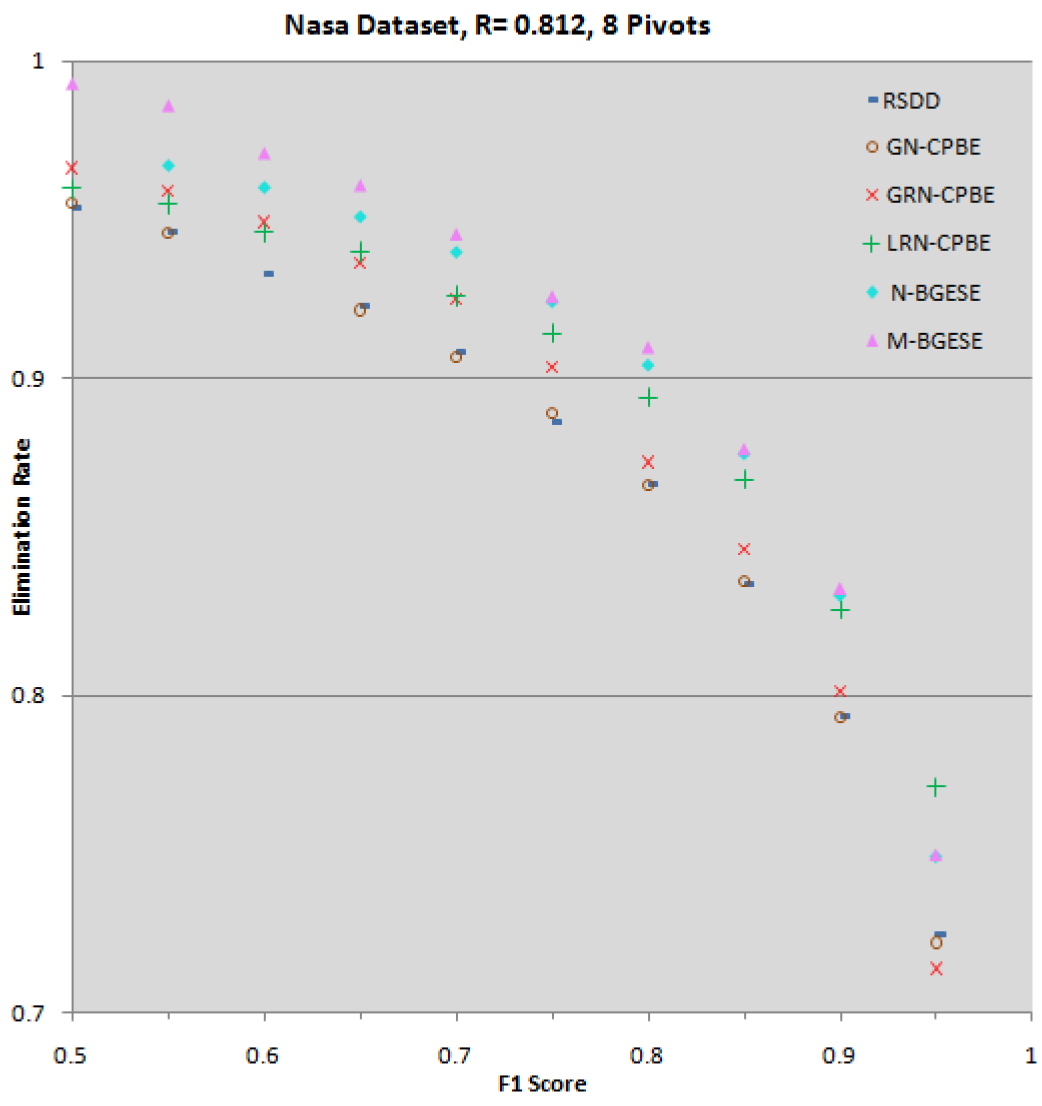


Figure 5.6: Elimination Rate-F1 Score comparison for dataset=Nasa, radius coverage=5%, pivot #=8, and Index structure elimination=26.3%

Figure 5.7 below includes the results of the first experiment performed with Gaussian dataset. Note that elimination rates of RSDD and GN-CPBE algorithms are missing for F1-scores smaller than 80% and 85% respectively. The elimination rates for these F1-scores could not be achieved. LRN-CPBE and BGESE variants have very similar performances up to 85%, where the performance of BGESE variants decrease much more as it is the case with other datasets. Both of N-BGESE and M-BGESE are outperformed by RSDD for the last two F1-scores. GN-CPBE's performance is very close to RSDD's as usual. Finally, GRN-CPBE's performance is particularly poor for all F1-scores.

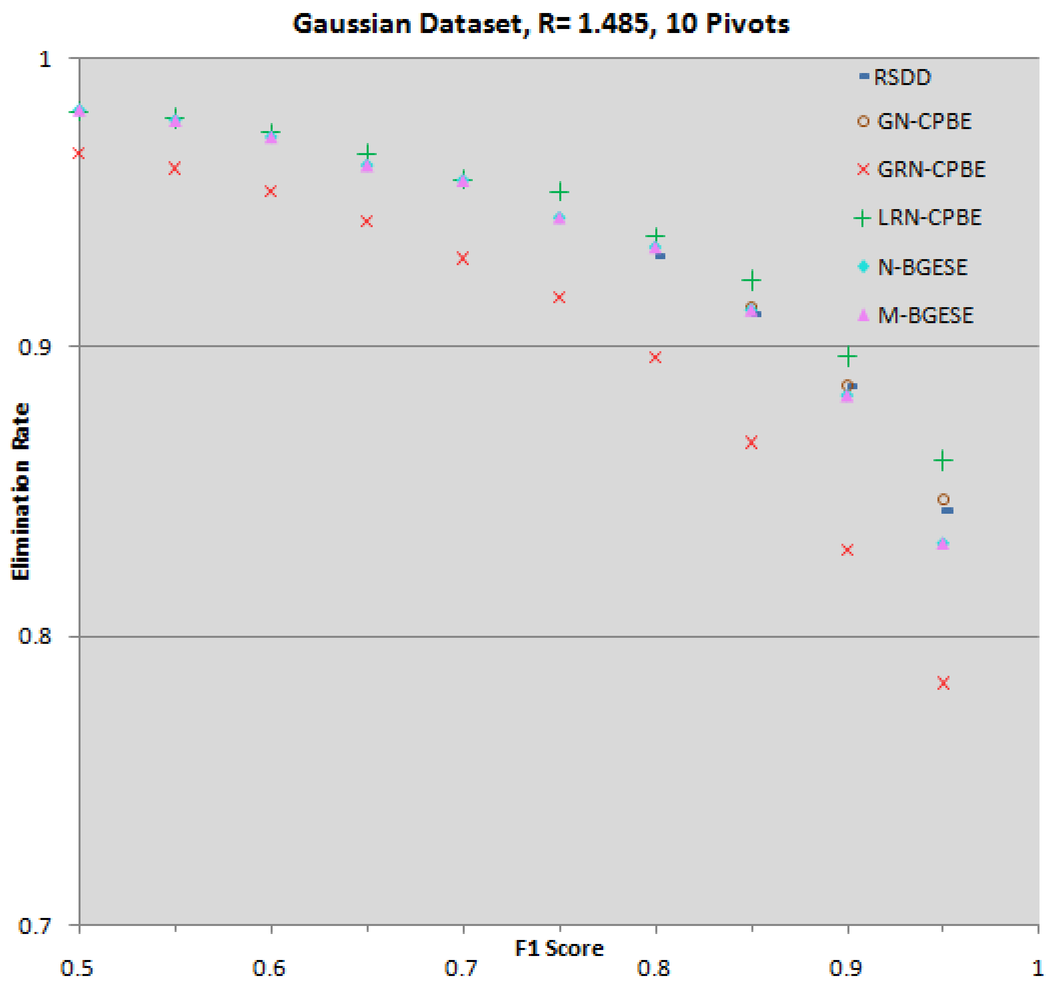


Figure 5.7: Elimination Rate-F1 Score comparison for dataset=Gaussian, radius coverage=1%, pivot #=10, and Index structure elimination=51.3%

Figure 5.8 illustrates the performance of the algorithms when the radius value is increased so as to cover 3% of all the database objects. The elimination rates of RSDD and GN-CPBE algorithms are again missing this time for F1-scores smaller than 70% and 80% in order. The performance of every algorithm with respect to each other is very similar to the corresponding performances depicted in Figure 5.7. The only exception is relatively improved performances of BGESE variants, where they are only outperformed by RSDD for just 95% F1-score.

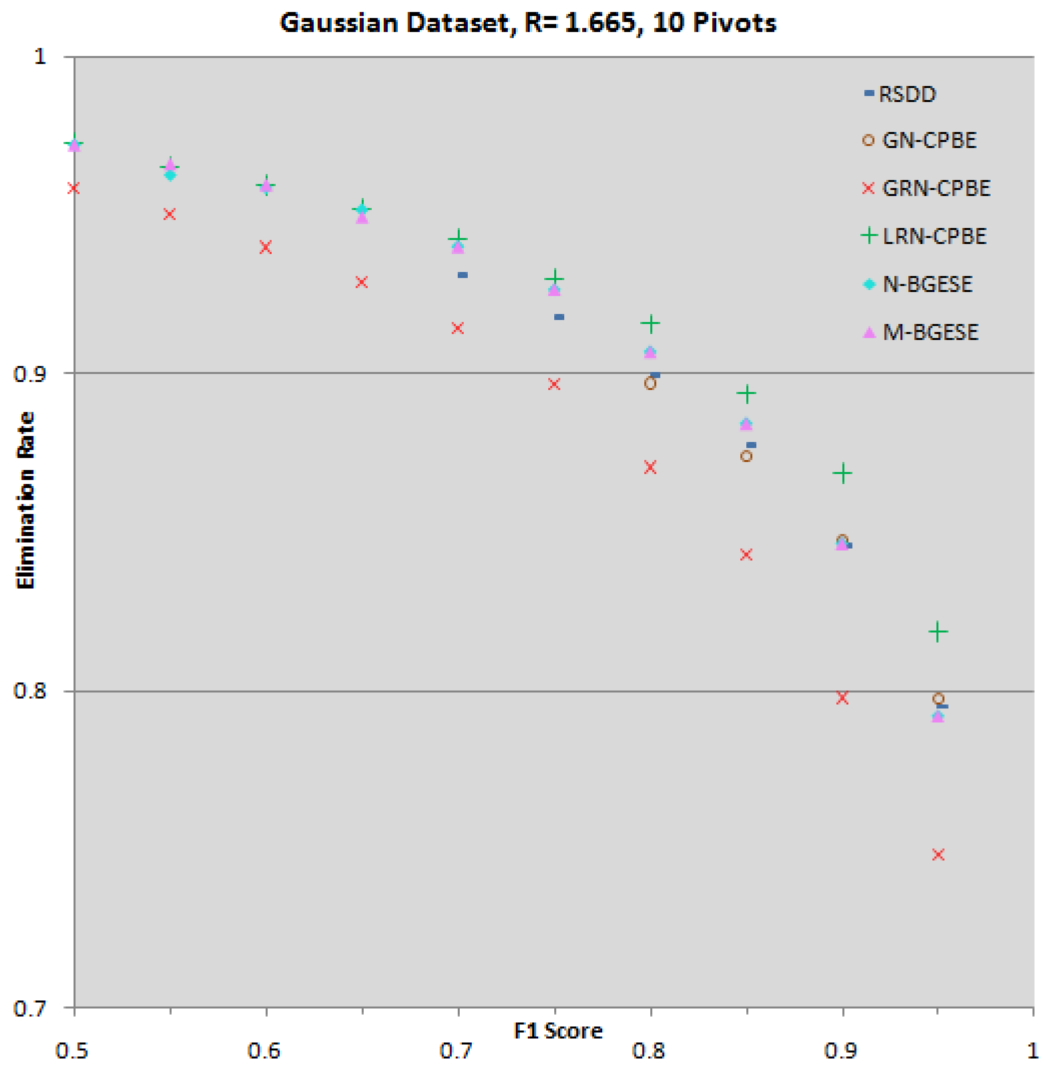


Figure 5.8: Elimination Rate-F1 Score comparison for dataset=Gaussian, radius coverage=3%, pivot #=10, and Index structure elimination=42.2%

Figure 5.9 includes the results of the last experiment performed with an increased radius to cover 5% of the objects in Gaussian dataset. The relative performance of the algorithms with respect to each other has not changed drastically. LRN-CPBE is still the best performer; where GRN-CPBE gives the worst performance. BGESE variants are relatively more successful, since they surpass RSDD for every F1-score value.

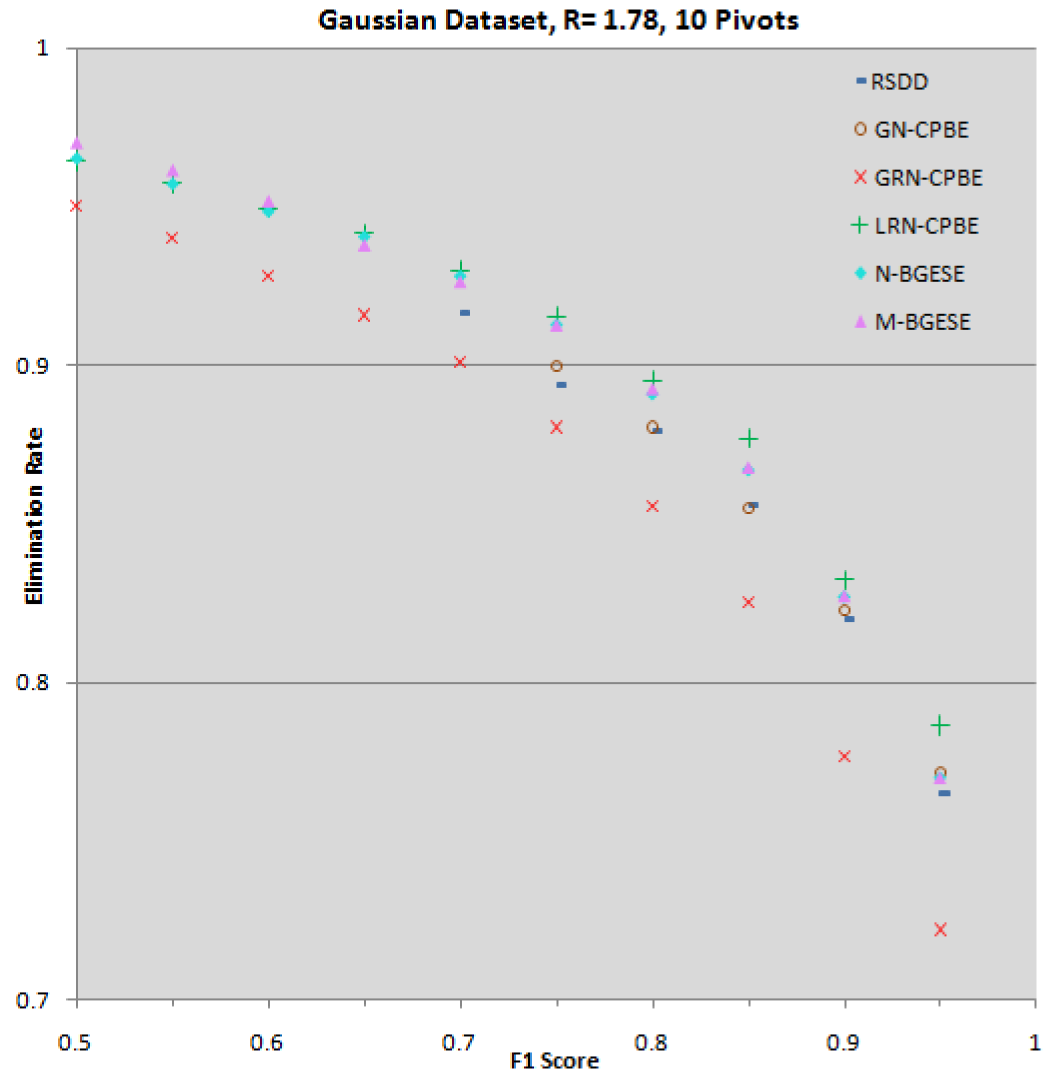


Figure 5.9: Elimination Rate-F1 Score comparison for dataset=Gaussian, radius coverage=5%, pivot #=10, and Index structure elimination=36.7%

5.3 Overall Comparison

We performed experiments with three different datasets for increasing radius values with radius coverage rates of 1%, 3%, and 5%. This section includes an overall comparison of the algorithms and a discussion about the variance of the performance with respect to dataset and radius value.

All of the proposed algorithms give higher elimination rates for lower F1-scores as mentioned. The effect of increasing the F1-score varies for different algorithms. BGESE algorithms seem to be the most effected ones, since they are surpassed by the base algorithm, RSDD, for a few higher F1-score values for the experiments performed with smaller radius values. Even if this is the case, BGESE variants outperform RSDD for most of the cases and they are particularly more successful as the radius value increases. This is due to the following reason: BGESE algorithms estimate the actual distance by using a model derived by the application of regression technique. In this approach, the overall error between the estimated and the actual distance is tried to be minimized. This means that the distances around the median are given more care than the distances at the edge of the distribution, so we think that regression models should give optimum performance around the median of the distance distribution. Therefore as we increase the radius value to approach the median, the performance of BGESE variants should increase as expected.

Regarding the comparison of BGESE variants between themselves, M-BGESE performs slightly better than N-BGESE for higher radius values and lower F1-scores. Other than that configuration, their performances are alike. The reason is that, M-BGESE eliminates some of the objects, which reside inside the radius of the query object and cannot be eliminated by N-BGESE. Since the number of such objects is very small with respect to the size of the dataset (1%, 3%, and 5% for our experiments), M-BGESE's performance becomes identical to N-BGESE's when a higher F1-score value is aimed.

LRN-CPBE seems to be the most successful algorithm in average. Although it is sometimes outperformed by one or both of BGESE variants, it shows a better performance than all the remaining ones for every dataset and radius configuration. This is not a big surprise considering the fact that LRN-CPBE provides a more sensitive guidance for the approximation of the distance of query object with the database objects with respect to the query object and the dataset.

GRN-CPBE is the algorithm that showed the most variance with respect to the radius and the dataset. It resulted in a poor performance especially for the Gaussian dataset; whereas it was slightly better than RSDD for Corel and somewhere in between for Nasa. This indicates a weakness with the algorithm. It might be the case that its performance is very sensitive to the distance distribution and distribution of the lower bounds of the particular dataset.

GN-CPBE has always given a very similar performance to RSDD. The reason is that the algorithm relies on *F(r) - F(l)* formula; but the cumulative distance distribution for the lower bound value is very small when compared to that of radius for most cases. As a result, the lower bound did not have much of an impact on the decision of the algorithm.

Chapter 6

Conclusion

In this thesis, we investigated approximation algorithms which can provide dynamic and particular guidance for the distance between a query object and a database object in metric spaces. Such guidance is based on various information derived from the dataset of interest and the lower and upper bound values calculated for the distance between a query object and the database object obtained via triangular inequality. Our contribution is the investigation and demonstration of the benefit of such a dynamic guidance; where most of the existing approximate similarity search algorithms are heavily dependent on the underlying index structure and offer index structure focused improvements.

We developed five different approximation algorithms; GN-CPBE, GRN-CPBE, LRN-CPBE, N-BGESE, and M-BGESE; which are dataset sensitive in their guidance for the approximation of distances. We compared the performance of these algorithms with another algorithm we developed: RSDD. We accepted RSDD as the base algorithm since it is only dependent on the distance distribution of the particular dataset; whereas the others are more sensitive to the query object.

Among the algorithms proposed, N-BGESE, M-BGESE, and LRN-CPBE perform better than RSDD nearly in most of the experiments. These three algorithms are also the algorithms we have expected to perform better than the others. The remaining algorithms could not provide an improvement over RSDD. This indicates some weakness related with these algorithms, which deserve further analysis as indicated below in Section 6.1.

6.1 Future Work

Although we have obtained improvements with some of the algorithms we developed, there is still room for further investigation and future work that should be carried out to see the benefits of our claims.

1. The number of pivots to use for the underlying index structure is determined in an emprical way. Further study can be carried out to determine the number of pivots with respect to the size and characteristics of the dataset. Moreover, the pivot selection mechanism explained in Kvp structure [8] might be used.

2. For LRN-CPBE algorithm, the *precision* value explained in Section 3.3 needs to be analyzed for an optimal value. A clever method to determine such a value needs to be determined instead of trying to discover it with an empirical way.

3. BGESE variants performance has been better for increased radius values. In order the algorithms to perform better for smaller radius values, we can develop multiple estimation models obtained via regression techniques instead of relying on a single global model. The construction of such models can be performed in a similar way to the construction of local distance distributions in Section 3.3.

4. The performances of GN-CPBE and GRN-CPBE algorithms have not been as satisfactory as expected. Although some of the possible reasons are explained in Section 5.3 above, further investigation might reveal the cause of the weaknesses of these algorithms.

5. Further work can be carried on the application of the finding of this thesis on different similarity search index structures.

6. The focus of this thesis has been range queries. We can develop similar approximate similarity search algorithms to answer k-nearest neighbor queries approximately. One perspective on approximating these queries is to put the objects or sets of objects (such as sub-trees) in a proper order so that the more likely candidates appear in front of the list. If we visit the objects on such an order, the distance of the k'th closest neighbor decreases more rapidly, increasing the chances of pruning of other objects. Ordering on the lower bound in particular gives the optimum solution in terms of the number of distances to the k-nearest neighbor queries. A good ordering can also be useful in an approximation scheme, if we can rapidly find out close neighbors, we can terminate the search early. In particular, our regression approach can be used to order the objects instead of using lower bound values only.

Bibliography

[1] G. Amato, F. Rabitti, P. Savino, and P. Zezula. Region proximity in metric spaces and its use for approximate similarity search. *ACM Trans. Inf. Syst.*, 21:192–227, April 2003.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. ACM*, 45:891–923, November 1998.

[3] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval.* ACM Press, New York, 1999.

[4] K. P. Bennett, U. Fayyad, and D. Geiger. Density-based indexing for approximate nearest-neighbor queries. In *ACM SIGKDD Conference Proceedings*, pages 233–243. ACM Press, 1999.

[5] T. Bozkaya and M. Ozsoyoglu. Indexing large metric spaces for similarity search queries. *ACM Transactions on Database Systems*, 24:361–404, 2002.

[6] S. Brin. Near neighbor search in large metric spaces. In *21th International Conference on Very Large Data Bases (VLDB 1995)*, 1995.

[7] C. Celik. Priority vantage points structures for similarity queries in metric spaces. In H. Shafazand and A. Tjoa, editors, *EurAsia-ICT 2002: Information and Communication Technology*, volume 2510 of *Lecture Notes in Computer Science*, pages 256–263. Springer Berlin / Heidelberg, 2002.

[8] C. Celik. *New Approaches to Similarity Searching in Metric Spaces.* PhD thesis, Department of Computer Science, University of Maryland, USA, 2006.

[9] E. Chávez, J. L. Marroquín, and R. Baeza-Yates. Spaghettis: An array based algorithm for similarity queries in metric spaces. In *Proceedings of the String Processing and Information Retrieval Symposium & International Workshop on Groupware*, SPIRE '99, pages 38–, Washington, DC, USA, 1999. IEEE Computer Society.

[10] E. Chávez, J. L. Marroquín, and G. Navarro. Fixed queries array: A fast and economical data structure for proximity searching. *Multimedia Tools and Applications*, 14:113–135, 2001. 10.1023/A:1011343115154.

[11] E. Chávez and G. Navarro. Probabilistic proximity search: fighting the curse of dimensionality in metric spaces. *Inf. Process. Lett.*, 85:39–46, January 2003.

[12] E. Chávez, G. Navarro, R. Baeza-Yates, and J. L. Marroquín. Searching in metric spaces. *ACM Comput. Surv.*, 33:273–321, September 2001.

[13] J. Chen, C. A. Bouman, and J. C. Dalton. Hierarchical browsing an d search of large image databases. *IEEE Transactions on Image Processing*, 9:442–455, 2000.

[14] P. Ciaccia, P. M., and Z. P. A cost model for similarity queries in metric spaces. In *In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'98*, pages 59–68, 1998.

[15] P. Ciaccia and M. Patella. Using the distance distribution for approximate similarity queries in high-dimensional metric spaces. In *Proceedings of the 10th International Workshop on Database & Expert Systems Applications*, DEXA '99, pages 200–, Washington, DC, USA, 1999. IEEE Computer Society.

[16] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. *Processing*, Athens, GR:426–435, 1997.

[17] Corel Image Features. `http://kdd.ics.uci.edu/databases/CorelFeatures/CorelFeatures.html/`, Accessed in August 2011.

[18] C. Faloutsos, W. Equitz, M. Flickner, W. Niblack, D. Petkovic, and R. Barber. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[19] C. Faloutsos and K.-I. D. Lin. Fastmap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. pages 163–174, 1995.

[20] J. Goldstein and R. Ramakrishnan. Contrast plots and p-sphere trees: Space vs. time in nearest neighbour searches. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 429–440, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[21] X. Lu, Y. Wang, and A. Jain. Combining classifiers for face recognition. In *Multimedia and Expo, 2003. ICME '03. Proceedings. 2003 International Conference on*, volume 3, pages III – 13–16 vol.3, july 2003.

[22] D. Maio and D. Maltoni. A structural approach to fingerprint classification. In *Proceedings of the International Conference on Pattern Recognition (ICPR '96) Volume III-Volume 7276 - Volume 7276*, ICPR '96, pages 578–, Washington, DC, USA, 1996. IEEE Computer Society.

[23] M. L. Micó, J. Oncina, and E. Vidal. A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recogn. Lett.*, 15:9–17, January 1994.

[24] Metric Space Library. `http://sisap.org/Metric_Space_Library.html/`, Accessed in August 2011.

[25] M. Patella and P. Ciaccia. The many facets of approximate similarity search. In *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on*, pages 308 –319, april 2008.

[26] E. V. Ruiz. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recogn. Lett.*, 4:145–157, July 1986.

[27] M. Tasan and Z. M. Özsoyoglu. Improvements in distance-based indexing. In *Statistical and Scientific Database Management*, pages 161–170, 2004.

[28] C. Traina, A. Traina, B. Seeger, and C. Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *In 7th International Conference on Extending Database Technology (EDBT*, pages 51–65. Springer-Verlag, 2000.

[29] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40:175–179, 1991.

[30] M. S. Waterman. *Introduction to computational biology.* Chapman & Hall, 1995.

[31] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[32] Weka 3 - Data Mining Open Source Learning Software in Java. `http://www.cs.waikato.ac.nz/ml/weka/`, Accessed in August 2011.

[33] P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with m-trees. *The VLDB Journal*, 7:275–293, December 1998.