EFFICIENT K-NEAREST NEIGHBOR QUERY PROCESSING IN METRIC SPACES BASED ON PRECISE RADIUS ESTIMATION

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Can Şardan

August, 2009

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(Supervisor)

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. A. Aydın Selçuk(Co-Supervisor)

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. İbrahim Körpeoğlu

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. Özcan Öztürk

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. Çetin Ürtiş

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

EFFICIENT K-NEAREST NEIGHBOR QUERY PROCESSING IN METRIC SPACES BASED ON PRECISE RADIUS ESTIMATION

Can Şardan
M.S. in Computer Engineering
Supervisor: Dr. Cengiz Çelik
Co-Supervisor: Asst. Prof. Dr. A. Aydın Selçuk
August, 2009

Similarity searching is an important problem for complex and unstructured data such as images, video, and text documents. One common solution is approximating complex objects into feature vectors. Metric spaces approach, on the other hand, relies solely on a distance function between objects. No information is assumed about the internal structure of the objects, therefore a more general framework is provided. Methods that use the metric spaces have also been shown to perform better especially on high dimensional data.

A common query type used in similarity searching is the range query, where all the neighbors in a certain area defined by a query object and a radius are retrieved. Another important type, k-nearest neighbor queries return k closest objects to a given query center. They are more difficult to process since the distance of the k-th nearest neighbor varies highly. For that reason, some techniques are proposed to estimate a radius that will return exactly k objects, reducing the computation into a range query. A major problem with these methods is that multiple passes over the index data is required if the estimation is low.

In this thesis we propose a new framework for k-nearest neighbor search based on radius estimation where only one sequential pass over the index data is required. We accomplish this by caching a short-list of promising candidates. We also propose several algorithms to estimate the query radius which outperform previously proposed methods. We show that our estimations are accurate enough to keep the size of the promising objects at acceptable levels.

Keywords: Similarity Searching, K-Nearest Neighbor, Metric Spaces.

ÖZET

METRİK UZAYLARDA İYİ BİR ALAN TAHMİNİ İLE EN YAKIN K KOMŞU SORGUSU İŞLEME

Can Şardan
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Dr. Cengiz Çelik
Tez Yöneticisi: Asst. Prof. Dr. A. Aydın Selçuk
Ağustos, 2009

Resim, görüntü, metin dökümanları gibi karmaşık ve düzensiz yapılarda, benzerlik taraması önemli bir işlemdir. Sıkça kullanılan bir yöntem, bu karmaşık verileri öznitelik vektörleriyle temsil etmektir. Bir başka çözüm ise, sadece bir mesafe fonksiyonuna dayanan metrik uzaylar yaklaşımını kullanmaktır. Objelerin iç yapıları hakkında herhangi bir bilgiye bağlı olunmadığından, daha genel bir iskelet oluşturulmaktadır. Metrik uzay yapısını kullanan yöntemlerin, özellikle yüksek boyutlarda daha iyi performans sergiledikleri gösterilmiştir.

Benzerlik taramasında kullanılan yaygın bir sorgu şekli, sorgu objesinin, verilen belirli bir alan içindeki komşularının bulunduğu, alan sorgusudur. Bir başka önemli sorgu ise, en yakın k komşu sorgusudur. İstenilen en uzak komşunun mesafesi değişkenlik gösterdiği için, bu sorguları işlemesi daha zordur. Bu nedenle, tam olarak k tane objeyi kapsayacak bir alan tahmini ile işlem bir alan sorgusuna indirgenebilir. Bu tekniği kullanan yöntemlerle ilgili genel bir sorun, alan tahminin düşük çıktığı durumlarda, algoritma az sayıda obje döndürür ve kalan komşuları bulmak için dizin verisi üzerinde birden çok tarama gerekir.

Bu tezde, en yakın k komşu taraması için, alan tahminine dayalı yeni bir sistem sunulmaktadır. Bu sistemde, sadece bir sıralı dizin taraması uygulanmaktadır. Bu, eksik komşu bulunduğu durumlar için, uygun aday olabilecek objelerin kısa bir listede tutulması ile sağlanmaktadır. Ayrıca, daha önce savunulmuş yöntemlerden daha iyi bir alan tahmini içeren yeni algoritmalar önerilmiştir. Bu tahminlerin, bahsedilen aday listesinin boyutunu düşük seviyede tutabilecek kadar gerçeğe yakın olduğu gösterilmektedir.

Anahtar sözcükler: Benzerlik Taraması, En Yakin K Komşu, Metrik Uzaylar.

Acknowledgement

I want to express my gratitude to my supervisor Dr. Cengiz Çelik for his inspiration and trust throughout this thesis.

I would like to thank committee members Asst. Prof. Dr. A. Aydın Selçuk, Asst. Prof. Dr. İbrahim Körpeoğlu, Asst. Prof. Dr. Özcan Öztürk, Asst. Prof. Dr. Çetin Ürtiş for reading and commenting on this thesis.

I am grateful to my family, for their support all through my life.

To My Family

Contents

List of Figures

Chapter 1

Introduction

In computer science, many applications use database management techniques in order to store and retrieve desired data. Complex, unstructured data requires a modeling phase so that it can be represented in some form that can easily be maintained by indexing. For instance, large sized images are generally transformed into feature vectors that hold some sort of information about them, such as color, texture, etc... Text documents are also represented as vectors; each dimension in the vector corresponding to a term in the document. In these vector spaces, the similarity between objects is defined by using geometric distance functions like the Euclidean distance. Although a large number of index structures are based on this framework, they lose their effectiveness in higher dimensions. Consequently, a simple, more general framework is developed as an alternative, known as the metric space model.

1.1 Metric Space

A metric space is defined by a set of objects O, and a distance function d between pairs of objects that satisfies the following properties:

non-negativity:

$$\forall o_1, o_2 \in O, d(o_1, o_2) \geq 0$$

symmetry:

$$\forall o_1, o_2 \in O, d(o_1, o_2) = d(o_2, o_1)$$

triangle inequality:

$$\forall o_1, o_2, o_3 \in O, d(o_1, o_3) \leq d(o_1, o_2) + d(o_2, o_3) \tag{1.1}$$

Metric space model provides a high level of abstraction, capturing a high variety of applications of similarity searching. It does not need to have any information about the internal structure of the objects, only a distance function that computes the similarity between them is sufficient.

1.2 Similarity Queries

In similarity searching, objects of a set O is classified based on the similarity criteria, which is defined by the distance between the object o and the given query object q.

A range query $R(q, r)$ is defined as:

$$R(q, r) = o \in O : d(q, o) \leq r$$

Every object within a distance of r to the query object q is retrieved. The query object itself is not included in the set of objects to be searched. One example of a range query can be: List employees with work experience less than 5 years. A derivation of the range query is defined by not only an upper limit but also a lower limit, where objects with distances in between are retrieved, such as: List employees with 2-5 years of work experience. In Figure 1.1, the objects n_1, n_2, n_3 are neighbors of q within r.

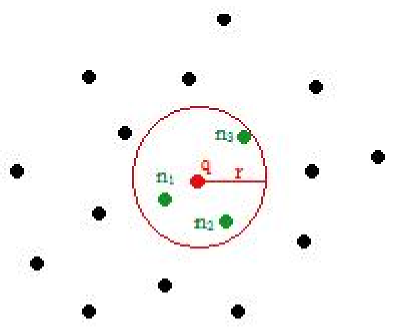


Figure 1.1: *Query Radius.*

A k-nearest neighbor query $kNN(q,k)$ is defined as:

$$KNN(q,k) = N \subset O, o_1 \in N, o_2 \in O\text{-}N, |N| = k : d(q,o_1) \leq d(q,o_2)$$

Unlike the range query, the number of neighbors k to be retrieved is specified in the k-nearest neighbor query. The objects are processed in such a way that, a list of neighbors N, of size k, is stored and updated whenever an object closer to the query than the farthest object in N is found, in which case that farthest object is removed. An example for k-nearest neighbor queries can be: List 3 employees having the most least experience. In Figure 1.1, the objects n_1, n_2, n_3 are the 3-nearest neighbors of q. Notice that, the two query examples retrieved the same elements, with different input parameters.

The triangular inequality property of the distance function in Equation 1.1 states that, the distance between two objects is related to their distances to a third object. Using this fact, metric access methods select a group of objects from the database and the rest of the dataset is indexed using these representative or pivot objects.

1.3 Indexing Methods

A popular approach used for indexing is to store data objects in a hierarchical way using tree structures. A typical tree node stores a subset of elements centered around a representative object. A covering radius defines the maximum distance of an object from this set to the representative. This information is used for defining distance limits between the query object and the elements in the node. During a range query, elements that stay out of these limits are said to be eliminated from the search space.

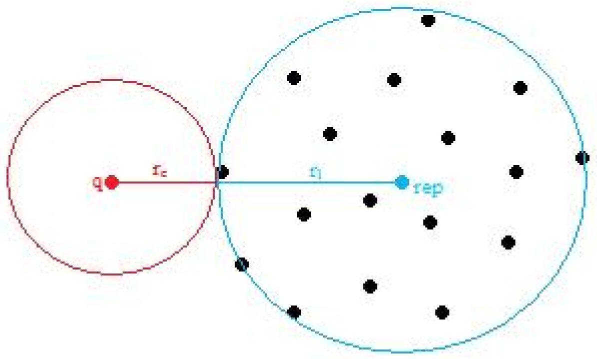


Figure 1.2: *Limiting Radius and Covering Radius.*

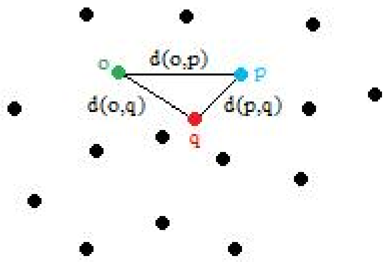


Figure 1.3: *Distance Bounding.*

Another way of indexing is to store objects according to their distances to a group of pre-selected pivot objects. These pre-computed distances are used together with the query object's distance to the pivot to identify whether a data object is in range of the query object. This technique used for eliminating objects is also referred as pivoting.

Besides tree-structures, there are some other approaches that index data using distance matrices. The pre-computed distances between objects and pivots are stored in array structures and used during query processing. A brief description about index structures recently developed will be presented in the next chapter.

1.4 Radius Estimation

In cases of high dimensional or very complex data, the cost of distance computation is the most important measure for evaluating query performance. For that reason, pruning abilities of metric access methods are very important.

Almost all existing indexing methods perform the technique of eliminating objects using the triangular inequality property of metric spaces, as described in the previous section. Since the maximum distance of an object from the query is known in advance, range queries are performed easily using these pruning abilities. In k-nearest neighbor queries, however, the radius covering the k closest object is not available beforehand. If this distance can be estimated accurately, k-nearest neighbor queries can be reduced to range queries and their performance can improve significantly.

Recently, different approaches are developed for estimating k nearest neighbor distances. Among these are methods that use histograms of distances between objects. In [14], the k nearest neighbors for each pre-selected pivot object are found and their distances are stored in a histogram. These are then used to define an upper bound radius estimate for query object's k nearest neighbors. Another approach uses a histogram of distances between every pair of objects in a set [21]. A probability density function is then created from this histogram and used for defining the number of object pairs within a given distance. This information is than used to estimate a k-nearest neighbor radius.

It has been argued that the distribution of pair-wise distances shows self similarity which means that the properties of the whole dataset is preserved similarly in parts of the dataset. A radius estimation based on this intrinsic dimensionality of datasets is presented in [28].

The performance of a k-nearest neighbor search depends on the overall cost of the query. The number of distance computations required is considered to be the most important evaluator, since it is difficult to calculate the distance between complex data objects. Construction cost, space requirements and CPU overhead are other significant issues that we will mention when comparing different approaches. A detailed explanation of described estimation methods are presented in Chapter 3, together with their overall cost analysis.

In this thesis, we propose several methods for estimating the k^{th} nearest neighbor distance. Our main focus is the precision of estimation and its affect on the performance of the query. We will present detailed comparison of our methods with related approaches in terms of estimation accuracy and disk usage. We will also address the case of underestimation when not enough number of neighbors are returned by the estimated radius. We will show that we outperform other approaches in such cases without the need of another estimation or scan of index data.

The organization of the rest of the thesis is as follows: In Chapter 2, a brief description of common index structures is presented followed by definitions of k-nearest neighbor algorithms based on radius estimation in Chapter 3. Then, methods for estimating k^{th} nearest neighbor distance are proposed in Chapter 4, followed by their experimental results in Chapter 5. Finally, in Chapter 6, we present concluding remarks and future work.

Chapter 2

Index Structures

Recent work on similarity searching lead to development of various access methods based on different index structures. Most common approach is to use tree structures for indexing data. Many metric access methods are based on index trees. An alternative solution presented in other methods is to use distance matrices for storing distance information among objects.

Indexing methods are evaluated according to their query performance as well as space and construction requirements. Query performance is based on two main measurements: the number of distance calculations, and any additional computation required to process and evaluate these distance measures, referred to as computational overhead. We discuss recent approaches using different index structures.

2.1 Tree-Based Structures

In tree-based index structures the basic theme is to use a hierarchical decomposition of the space. One popular approach using tree structures is to partition data into clusters. The objects close to each other are grouped together in these clustering-based methods, where a single object, ideally located near the center of the group is used as the representative. Another approach used in tree structures is to define the partitions based on distance ranges to one or more pivots selected among them. Those with similar distances to these pivot objects are put inside the same subtree; however this does not necessarily indicate that they are also close to each other. The pivots are only used for objects within their subtrees, hence the name local pivots.

The *GNAT* [1] is an example for tree-based methods, using more than two representatives for a partition. Together with the radius of the region around the representative, the information of the minimum and maximum distances to the objects in every other subset is maintained. Compared to the common local-pivot based structure *vp-tree* [25], it requires fewer distance computations in exchange of higher construction cost. However, recent experiments [8, 6, 1] show that GNAT performs worse than distance matrix methods in query performance, while it needs less space and computational overhead.

The *M-tree* [10] and the *Slim-tree* [22] are disk-based structures which are very similar to the GNAT. In order to support and efficiently process dynamic operations they store less precise data than GNAT, resulting in poorer query performance.

The vp-tree [25] uses a single pivot and a branching factor k to divide objects in a node to k groups differentiated according to their distances to the vantage point. The node itself stores only the $k-1$ values defining the distance ranges for each subtree. The shortcoming of the vp-tree is that, for higher dimensions, objects tend to be at similar distances to the pivot, which eliminates the power of distinguishing objects. As an improved version, *mvp-tree* [2] uses more than one vantage points to further divide the partitions created by other vantage points.

2.2 Distance Matrix Methods

An alternative solution to tree-based structures is to use distance matrices for storing distances between pivots and objects in the dataset. Contrary to local pivot-based methods, each pivot is used in the processing of every object, therefore referred to as global pivots. During query time, these pre-computed distances are used to eliminate objects based on the concept of pivoting. Therefore, a distance computation is required only for those remaining candidate objects which have not been eliminated. With the requirement of higher space and construction time, the global pivot-based methods can boost up query performance by increasing the number of pivots. This is the main advantage of distance matrix methods, since the number of pivots are limited in tree structures decreasing their flexibility to provide enough elimination power especially in high dimensional distributions. In such distributions, there appears a large number objects that are at similar distances to both pivots and the query object. Nevertheless, in global-pivot based methods, as many number of pivots as needed can be used at the expense of higher space and construction cost.

AESA [27] is known to be the first method in which all the data objects also serve as pivots. In *LAESA* [18], a subset of objects are selected as pivots instead of all of them. The distances between these pivots and rest of the objects are stored in arrays to be used in query time. In the *Spaghettis* structure [7] , the computational overhead is reduced by sorting these stored distances to pivots and using a binary search among them. This requires, however, additional space and construction time. The *Fixed Query Arrays* (FQA) [6] eliminates this extra need of space by storing less precise distance values, which reduces the accuracy of pivots especially in high dimensional data.

The main trade-off in distance matrix methods is that, greater query performance can be achieved in exchange of higher space usage and construction time. A solution for this is proposed in the *Kvp* structure [5].

2.3 The Kvp Structure

It is shown that a pivot is more effective for objects that are close to or distant from it [5]. This is illustrated in Figure 2.1, where we observe that the number of

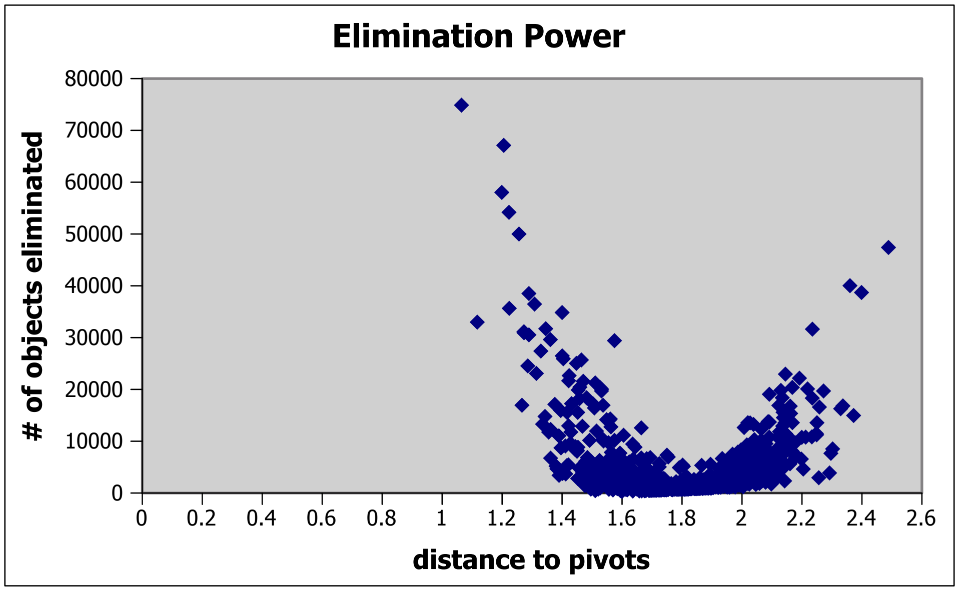


Figure 2.1: *Elimination power with respect to pivot-query distance.*

objects eliminated by a pivot according to its distance to a query object. In the construction phase, for each object, Kvp stores only the most promising pivot distances based on this information. In order to have effective pivots for every object in the dataset, the selection process is implemented in such a way that objects that are maximally separated from each other are chosen as pivot objects. The selection process is presented in Algorithm 1.

When the next pivot is to be selected, the minimum distances of objects to currently appointed pivots are computed. Then, the object with the maximum value of this distance is chosen as the next pivot. This ensures that selected pivots are distant from the general population, which provides better elimination power.

The prioritization of pivots in Kvp decreases the computational overhead and space requirement, since less number of distances are stored and used. Other than that the same pivoting technique is used similar to other global pivot-based methods.

Algorithm 1 Pivot Selection.

Input: set of objects O, number of required pivots n_P
Output: the set of pivots P of size n_P

```
define array $minDistances$ of size $n_O$, set all values to inf
select first object $o \in O$ as a pivot
add $o$ to $P$
while size of $P < n_P$ do
    set $max$ to 0
    set $p$ to last pivot selected
    for all $o \in O - P$ do
        compute d(o,p)
        if $d(o,p) < minDistance(o)$ then
            set $minDistance(o)$ to $d(o,p)$
        end if
        if $max < minDistance(o)$ then
            set $max$ to $minDistance(o)$
        end if
    end for
    select $o$ with $minDistance(o) = max$ as the next pivot
    add $o$ to $P$
end while
```

Chapter 3

K-Nearest Neighbor Algorithms

The simplest way of finding k-nearest neighbors of a given query object is to compute its distance to every object in the dataset. This will require n distance computations, where n is the total number of objects. Since this is not acceptable in cases where distance computation is relatively expensive and dominant in overall cost of a query, a lower bounding principle is incorporated in Algorithm 2.

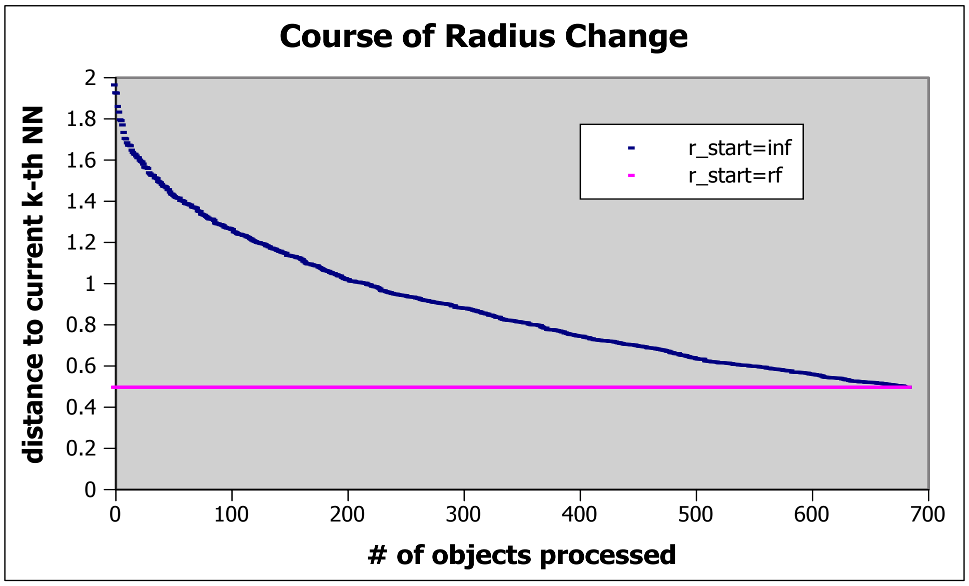


Figure 3.1: *The change of the distance to the k-th Nearest Neighbor as more objects are processed. The straight line shows the value r_f.*

The radius covering the k-nearest neighbors of the query is set to infinity at the beginning. The objects are processed in random order and eliminated according to their lower bound distances to the query object. During this operation, candidate objects are inserted in a list NN_q that contains current nearest neighbors of the query. After NN_q is filled with k neighbors, the radius r is set from infinity to the distance of the farthest object in the list. The process continues, updating r whenever a closer object than the current k^{th} neighbor in NN_q is found.

Algorithm 2 Basic KNN Algorithm. Object-pivot distances are pre-computed.

Input: set of objects O, set of pivots P, query q, number of neighbors k
Output: the k-nearest neighbors of q

```
compute and store distances d(p,q) $\forall p \in P$
set $NN_q$ = list of kNNs of size $k$
set $r = \infty$ (distance to $k^{th}$ NN)
for all $p \in P$ do
    if $d(p,q) < r$ then
        add $p$ to $NN_q$
        set $r = d_{max}NN_q$
    end if
end for
for all $o \in O$ do
    compute lower bound value for d(q,o), lb(o)
    if $lb(o) < r$ then
        compute distance d(o,q)
        if $d(o,q) < r$ then
            add $o$ to $NN_q$
            set $r = d_{max}NN_q$
        end if
    end if
end for
```

The shortcoming of this base algorithm is that, since the radius of the k-nearest neighbors starts from infinity, the process can not eliminate a good deal of objects in earlier stages of the process, resulting in high number of distance computations. An example of the change in average r values with respect to number of objects processed, starting from k, is shown in Figure 3.1.

The final r value, which we will call r_f, is the actual radius of the k-nearest neighbors of the query object. If this final radius r_f had been used to eliminate objects from the beginning, only objects with lower bounds to the query object less than r_f would have been processed. In other words, for each r value in the graph, objects with lower bounds between r_f and r could be eliminated on top of the objects that are eliminated by the basic algorithm.

A range query, given the distance to k^{th} nearest neighbor as the query radius, is therefore the best theoretical algorithm in terms of number of distance computations for finding k-nearest neighbors. An equivalent algorithm is presented in K-LAESA [19], which extends LAESA [18] to find k neighbors instead of only 1. The basic approach is to process objects in ascending order of their lower bound of distances to the query object. Candidate objects are added to the current neighbors list NNq. Whenever an object with a lower bound greater than the farthest object in NNq is found, the algorithm terminates. This way, only the objects with lower bound values smaller than the actual k-nearest neighbor radius are processed, which is exactly the same in the described range query. However, K-LAESA algorithm requires too much construction time and space as well as more than one full scan of index data for the sorting procedure; which also increases the CPU overhead.

A similar approach is also proposed in tree-based structures. An incremental ranking algorithm [13] processes subtrees using a global priority queue, which holds in sorted order the visited nodes and data objects in ascending order of distances, such that the front of the queue holds the element with the smallest distance to the query. The queue is processed in such a way that, if the front of the queue is a node, then its children are added to the priority queue. If it is a data object, it is added as the next nearest neighbor to the NNq. The algorithms stops when NNq is full with required number of neighbors, k. Only the nodes with distances smaller than the k^{th} nearest neighbor distance are traversed. The node distances are defined similar to the lower bound distances in distance matrix methods. Therefore, the overall cost is similar to the K-LAESA algorithm.

The main constraint for k-nearest neighbor queries is the unknown distance of the k^{th} nearest object. As pointed out, a range query would definitely decrease the number of distance computations, if the k-nearest neighbor radius could be estimated beforehand. This would also discard the need for a sorting procedure, significantly improving the overall performance of the query processing. For that reason several algorithms estimating the k-nearest neighbor radius are developed. A basic approach is described in [21], where the algorithm uses the information provided by the distance distribution of a dataset. The basic idea is forming histograms of pair-wise distances between objects. Then, this histogram is scaled and viewed as a probability density function as:

$$H(S) \Rightarrow P(S, r)$$

For a dataset of size n, with the requested number of neighbors being k, an estimate for the distance of the k^{th} nearest neighbor of the query object is then derived by the following formula.

$$n \int_0^{E(d(q,KNN(q)))} P_q(S, r)dr = k$$

The probability of finding an object at distance r of the query is computed for a range of values of r, setting the cumulative probability to be k/n. This task requires pre-computed distances among every object, increasing the construction cost significantly.

An extended approach from this method is to take a subset of the described histogram to form another, storing the distances of objects to their k^{th} nearest neighbors for all values of k that may be encountered in queries. Using this new histogram a probability density function is created likewise and used for estimating k-nearest neighbor radius for the query object. The following formula summarizes this approach, where $P(k, S, r)$ is the probability of the k^{th} neighbor to be at a distance of r to the query object.

$$E(d(q, KNN(q))) = \int_0^\infty rP(k, S, r)\,dr$$

Effectively, it is equivalent to taking the average k-th nearest neighbor distances of all the data objects. These two estimation methods are global; meaning that their estimation is same for every possible query object.

Another algorithm that uses histograms is presented in [14]. A small number of pivots are selected from the dataset and their distances to nearest neighbors of them are pre-computed and stored in a matrix. For a given query object, an upper bound to its k^{th} nearest neighbor is determined using the triangular inequality property as illustrated in the following formula.

$$r_{est} = \min_{1 \le i \le m} [d(q, p_i) + H(p_i, KNN(p_i))]$$

The equation implies that, for a certain value of k, the distance between the query object and a pivot added to the pre-computed k^{th} nearest neighbor distance of that pivot is definitely larger than the possible distance of the query to its k^{th} nearest neighbor. Therefore, the minimum upper bound value defined by m pivots is used as a local estimate. The pivots are selected among the objects minimizing the total value of the pivot distances.

The algorithm requires less space compared to the previous histogram based approaches, storing $p.k$ distances, where p is the number of pivots and k is the number of nearest neighbors. However we will show that it does not estimate an accurate radius which leads to an increased number of distance computations.

The $kNNF$ algorithm in [28] uses the intrinsic dimensionality of datasets for radius estimation. It is described that, the distance distribution between pairs of objects show a fractal behavior, or self similarity. This implies that, certain parts of the dataset shows similar properties to the whole set. Based on this idea, the number of pairs within a certain radius r is defined using the power law.

$$PC(r) = K_p + r^D \tag{3.1}$$

D is the correlational fractal dimension of the dataset [11], and K_d is a proportionality constant. Based on this relation, the radius of the k-nearest neighbors is estimated as follows: the logarithm of the constant K_p is derived from (3.1) using the total number of pairs in the dataset $n(n-1)/2$ for $PC(r)$, where n is the database size and R is the maximum distance between two objects.

$$K_d = \log(K_p) = \log(n(n-1)/2) - D\log(R) \tag{3.2}$$

The number of distances between a subset of k objects and the objects of the whole set N is defined as $n(k-1)/2$. For a required number of neighbors k, the radius is estimated using the relation in (3.2).

$$\log(n(k-1)/2) = \log(n(n-1)/2) - D\log(R) + D\log(r_f)$$
$$r_{est} = R^{((\log(k-1)-\log(n-1))/D)}$$

In case less than required number of neighbors is returned, the same relation is used for a local estimate, only replacing n with the number of retrieved objects $k' < k$, and R with the estimate r_{est}. This way instead of the whole dataset, only the density around the query object is considered and a more accurate estimation is made. The algorithm uses disk-based index structure slim-tree, as a consequence, this local estimation decreases the performance in terms of disk usage, since it requires another sweep of the index data. However, experiment results show that the local estimation is almost never used, because the global estimate, r_{est}, is greater than the actual k-nearest neighbor distance most of the time. This means, the algorithm over estimates the query radius in general, decreasing the number of objects eliminated, therefore increasing the number of distance computations. On the other hand, the algorithm does not require the additional space and construction time for pre-computed distances between objects prior to the query processing.

Chapter 4

Precise Radius Estimation

We have elaborated that a k-nearest neighbor query can be reduced to a range query using a radius estimate. The critical parameter in this transformation is the value of r to be used as an input. We have developed a number of methods for estimating this radius of the k-nearest neighbors of a query, described in the following sections. Their relative errors to the actual distances are discussed in the next chapter, together with the overall performance of the algorithms.

4.1 Global Estimation

A global pair-wise distance distribution is constructed using the pre-computed distances between a sample set of objects from the database. This approach is identical to the one described in [21], except the fact that only a subset of distances is used, decreasing the construction cost. Example distributions for different datasets are shown in Figure 4.1. A certain point on the curve defines the ratio of object pairs at a distance r to the total number of pairs in the dataset. The area under the curve therefore adds up to 1. Observe that the percentage of objects that are close to or distant from each other is relatively small compared to the rest of the dataset.

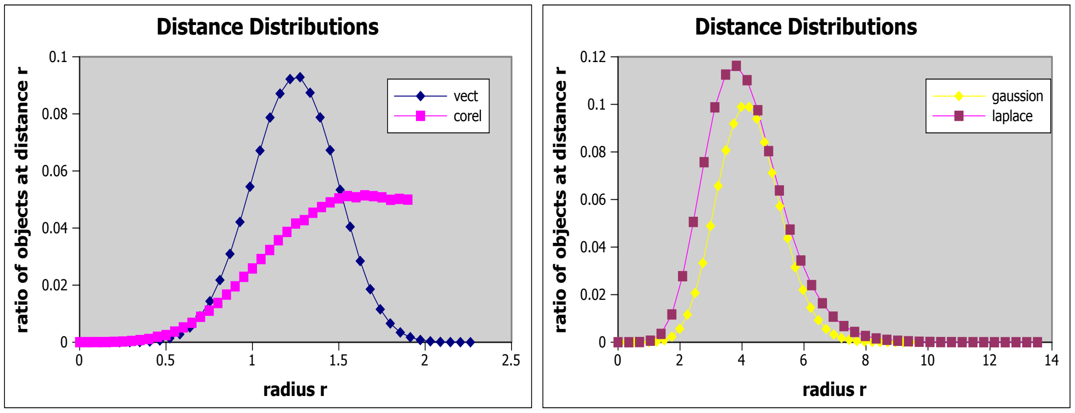


Figure 4.1: *Distance Distribution for different datasets, including uniform, gaussian and laplace distributions, and color histogram from the corel data.*

A global estimation is made using the distance distribution. A cumulative probability function is defined using the area under the curve for a range of r values.

$$F(r) = \frac{\textit{number of distances} \leq r}{\textit{total number of distances}}$$

The $F(r)$ function is used to find the number of objects in range r of a given query object q, the number of distances less than r between q and any data object o. The ratio that $F(r)$ returns is multiplied by the total number of objects at any distance to q, which is the size of the database n.

Consequently, the reverse of the function $F(r)$ estimates a radius for a given number of required neighbors k.

$$F^{-1}(\frac{k}{N}) = r \qquad (4.1)$$

A significant problem with global estimation is that it gives the same radius for any possible query object. Therefore by definition, although accurate on average, it will give an overestimate for half of the query objects, and an underestimate for the other half. The former will increase the number of distance computations, whereas in the latter case, not enough neighbors would be returned. For that reason, an estimation considering the query location is crucial in precision of the radius estimate. Even though global estimation is not adequate by itself, it is incorporated in some local estimation methods described in the following sections.

4.2 Local Estimation

A query object, due to its distinct location, may not abide by the general distribution of pair-wise distances. In that case, the location of the query object needs to be considered. One approach is to select a sample of objects from the dataset to be processed for a local estimation. Since this sample set is stored in the memory, this procedure is done efficiently, without the requirement of additional disk scans.

4.2.1 Progress Of Query Range

Regression is a technique used to model a numerical data consisting of values of a dependent variable, and one or more independent variables. It is commonly used for prediction. There are several types of regression for fitting (least squares fitting) a curve through a given set of points. The change of the dependent variable; distance of the current k^{th} nearest neighbor, along with the number of objects processed follows that of a power law distribution illustrated in Figure 3.1 on Page 12 defined by the function:

$$y = a \cdot x^b$$

The sample objects in memory are processed using the base algorithm on Page 13. The change of the radius value together with the number of objects processed is given as input to the regression function. Then, given the total number of objects in the dataset, the final radius is predicted, which is used as a local estimation of the query object.

A shortcoming of this method is that the objects processed in memory need to be reconsidered after the radius estimation. The reason is that they may or may not lay in the range r_{est} of the query. This is overcome by using an array storing the already computed distances of objects in the sample set. This eliminates the need of redundant distance re-computation for these objects.

4.2.2 Uniformity of Local Density

Due to the self similarity of the dataset, regardless of the total number of objects, the density around the query stays at similar levels. This suggests that if the distance of the k^{th} nearest neighbor among a subset of objects of size m is r for a given query, then the radius for $k * n/m$ nearest neighbors in the whole set will also be r. This is illustrated in Figure 4.2, where green points are sampled objects of size m=10, where n=20. Based on this observation, the sample objects in memory are processed using the base algorithm, finding the $k' = k * m/n$ nearest neighbors. The distance of the farthest object in this list is then used as a local estimation.

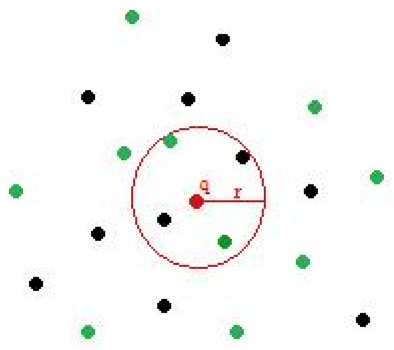


Figure 4.2: *Density around a query object. Green points refer to objects in the sample set. The whole dataset may include other objects within the radius, however the ratio of these objects to the total size is expected to stay the same.*

An important parameter for this method is, like in the previous method, the number of objects to be sampled and processed in memory. There are some serious restrictions on the sample size. For instance, if k is a small value such as 5, the sample must be at least one fifth of the whole dataset so that k' can be set to at least 1. This shortcoming is addressed in a modified version of this method, in cases when significant increase to the memory size is unavailable.

Projection of Distance Distribution

If the k value is very small, then it becomes impractical to have a meaningful value for k' while keeping the sample size small. For such cases, we propose a new method based on the observation that while the distance of the k^{th} neighbor is expected to be different for different query objects, it will be proportional for different values of the number of neighbors. For example, if the distance of the 5^{th} nearest neighbor is above average by 20%, then we also expect the distance of the 10^{th} nearest neighbor to be about 20% higher than the average. Based on this, whenever k' is too small, we will use another value k'' that will let us use less samples.

Recall the reverse cumulative probability function in Equation 4.1 on Page 19. The previous method states that for $k' = k * m/n$,

$$F^{-1}(\frac{k'}{m}) = F^{-1}(\frac{k}{n})$$

Another parameter, k'', is introduced in this method indicating the minimum number of neighbors to be retrieved among the sample objects in memory, whose radius will be used for estimation.

$$\frac{r(k')}{r(k'')} = \frac{F^{-1}(\frac{k}{n})}{F^{-1}(\frac{k''}{m})} \tag{4.2}$$

The left side in the Equation 4.2 illustrates the radius of the neighbors, k' and $k'' > k'$ respectively, among sample objects in memory. Since k' and therefore $r(k')$ is too small, $r(k'')$ is computed during memory processing. Using global estimations for both values, the radius of k nearest neighbors of the query in the whole set is estimated. Recall from previous method that $r(k')$ in m objects is expected to be similar to $r(k)$ in n objects.

The processing of the sample set of data objects increases the computational overhead, though it does not affect number of distance computations, since they are stored and reused during query processing with the estimated radius. Furthermore, considering they are processed in memory, efficiency can be improved by using the lower bound sorting algorithm described in [19]. The result of this modification is shown in Figure 4.3.

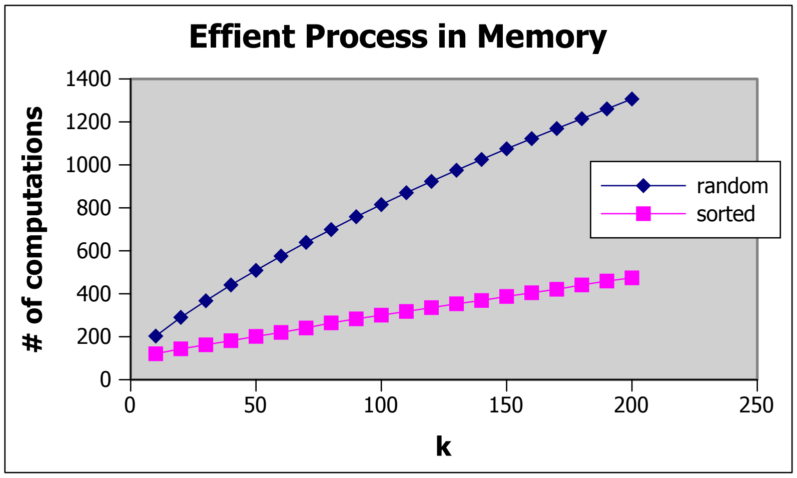


Figure 4.3: *The computational overhead for processing objects in the sample set of size = 10000, for different values of k.*

We will emphasize the effect of the sample objects size and the k'' value in the experimental results.

4.2.3 Static Pivots

As mentioned before, the location of the query object affects the expected radius of its k nearest neighbors. A second approach to identify the query whereabouts is by use of pivot distances. For instance, if an object is far away from the general population its neighbors will also be relatively farther than expected. Similarly, if it's too close to the set of objects, its radius will be smaller than expected.

In order to isolate the location of the query, its distance to a set of appointed pivot objects are computed. We expect those with a large value of average distance to pivots to have larger radius for its k nearest neighbors, and vice verse.

A learning phase is carried out based on the relationship between an object's distance to its pivots and k nearest neighbors. A sample set of objects are selected for training and processed observing the values, $d_{knn}(q)$ and $d_{avg}(q,p)$. A regression technique similar to the one described before is used for modeling the relation between these distance values. We observe from Figure 4.4 that there is almost a linear dependency in the relation.

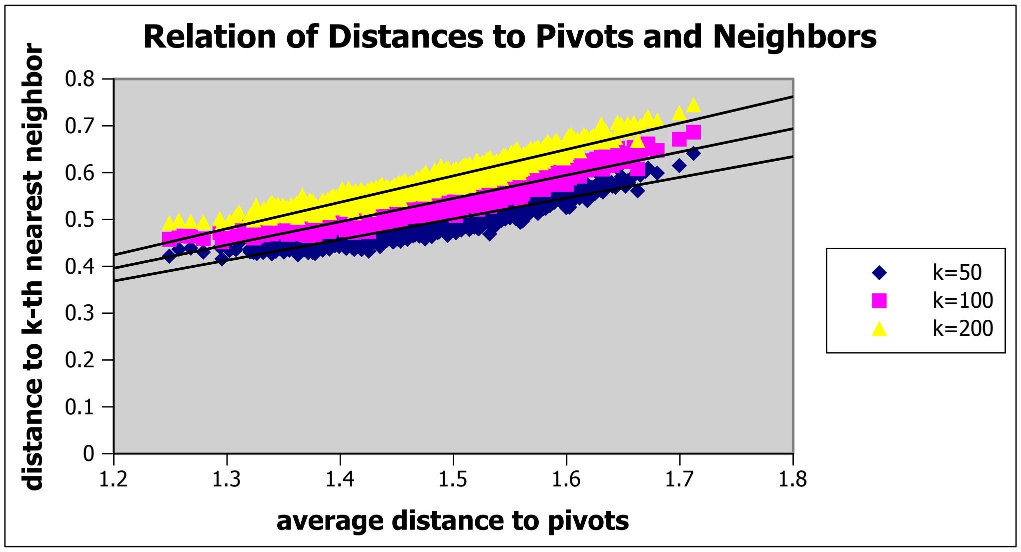


Figure 4.4: *K-nearest neighbor distances with respect to average pivot distances for objects in the training set of size = 1000.*

Prior to processing the objects in the dataset for determining the k nearest neighbors of a query, its pivot distances are computed. The average is then used for predicting the distance of the k^{th} nearest neighbor from the regression model, which is then used as a local estimation. We will illustrate the effect of the number of pivots appointed for use in the estimation.

Chapter 5

Experiment Results

Throughout the experiments, we have used several datasets, including uniform and non-uniform distributions. Along with the random vector sets with uniform, laplace and gaussian distributions of size 100k and dimension 10, we processed the color histogram of size 68040 and dimension 32 from the *Corel* data.

We have tested 100 samples of query objects for each dataset. Since average error values of the radius may not show the actual accuracy of the estimation, we used the term relative error in order to describe the effectiveness of the algorithms. Along with relative error, we present the number of distance computations, and the number of neighbors retrieved by the estimated radius to illustrate how much of the backup list of objects need to be processed to reach the requested number of neighbors k.

5.1 Global Estimations

Including the algorithms presented in [21], we tested the global estimation derived from the sample distance distribution. Recall that the first approach in [21] and our global estimation are similar except only use different sizes of samples. The errors of estimations are illustrated for different datasets in Figure 5.1.

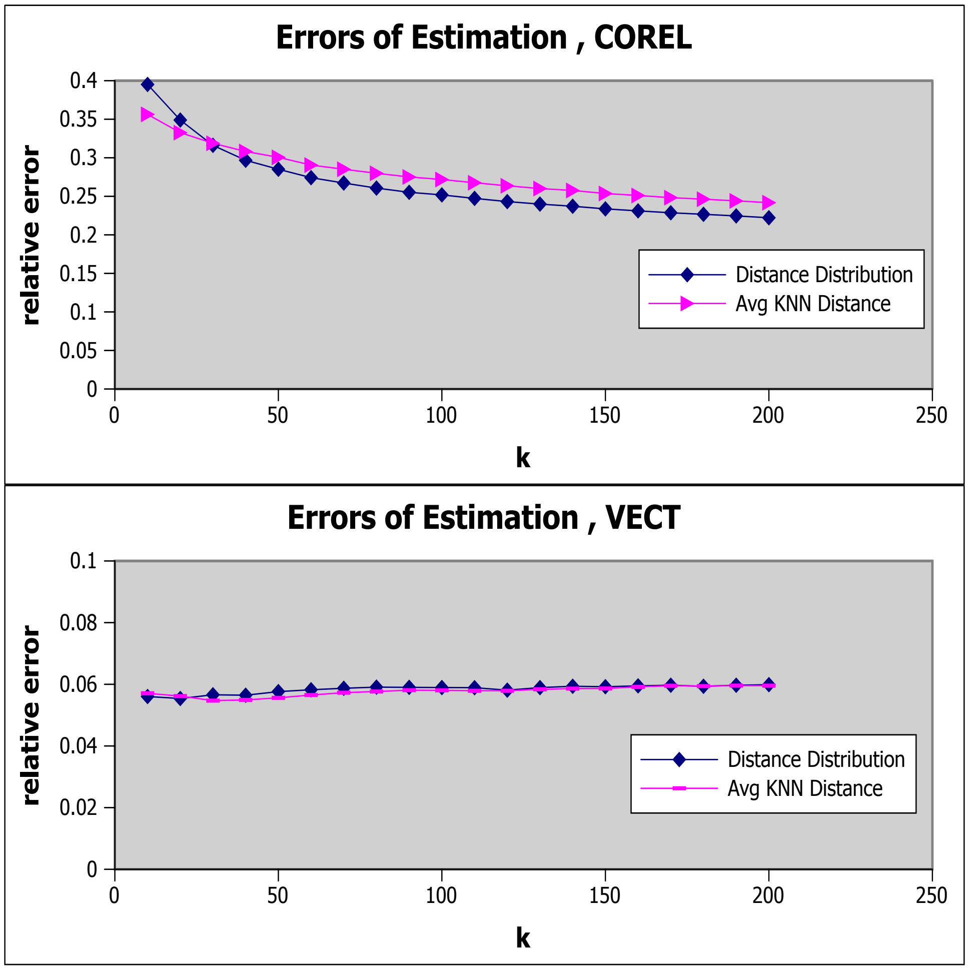


Figure 5.1: *Relative Errors of global estimations; for 'uniform vector' and 'corel' datasets.*

The key observation from the resulting graphs is that, especially for uniform data, a global estimate is very close to the actual radius of the neighbors in average. However, this does not necessarily mean that it is accurate for each specific query. Quite the contrary, the method overestimates for one half of the queries, and underestimate for the other half, returning less than required number of neighbors. The ratio of the number of neighbors returned to k is illustrated in Figure 5.2.

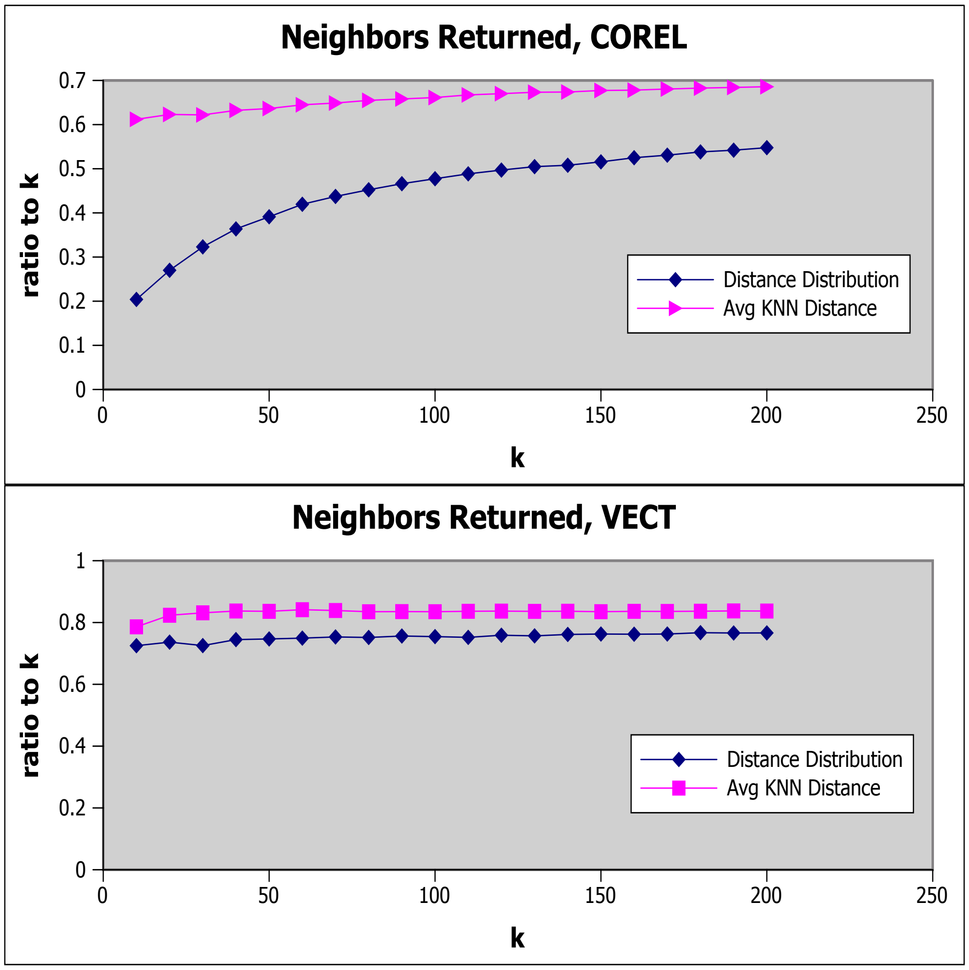


Figure 5.2: *Proportion of the number of neighbors returned by global estimations; for 'uniform vector' and 'corel' datasets.*

5.2 Local Estimations

As we described in the previous chapter, several techniques for identifying the location of the query is used in order to perform a local estimation. In exchange of increasing the computational overhead, all approaches give more accurate results compared to global estimation.

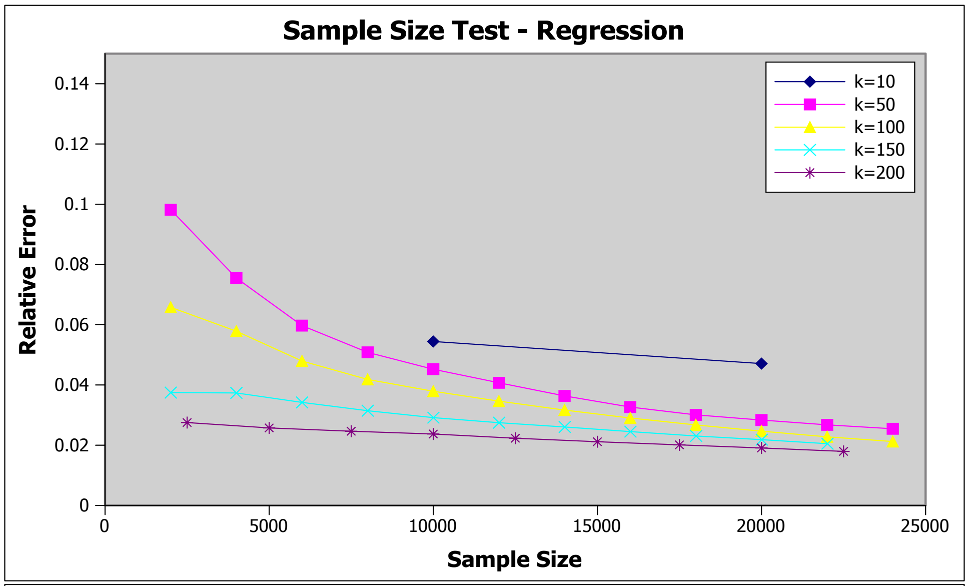
5.2.1 Progress of Query Range

A common approach used for prediction over a distribution of points is to use the regression model. An important parameter for this method is the number of independent variables used for the input data. This corresponds to the size of the sample set processed for estimation. The significance of this size is observed in Figure 5.3.

We see that in *corel* data, the effect of increasing the sample size descents at a slow rate. In uniform vectors however, larger sample size becomes irrelevant after a certain value, especially for bigger values of k. Since increasing sample size signifies additional computation, limiting it to a reasonable value is essential. Based on these observations, we define the sample set size to be a tenth of the total data size. Smaller values also perform well, especially in uniform data, however, we selected this value for fair comparison with other methods.

5.2.2 Uniformity of Local Density

The second idea is to process a sample set of objects and draw a conclusion about the density around the query. Recall that for a certain radius, although the number of neighbors of an object changes by increasing the total data size, its ratio to the whole set stays the same. Again, the size of the sample set is a key parameter. In Figure 5.4, the effect of the number of objects processed is represented in terms of relative error.



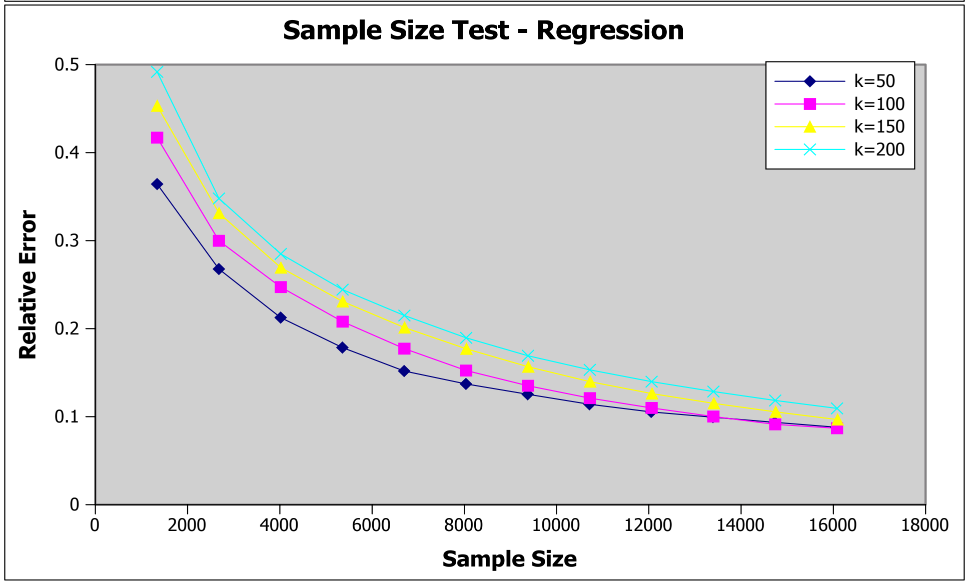
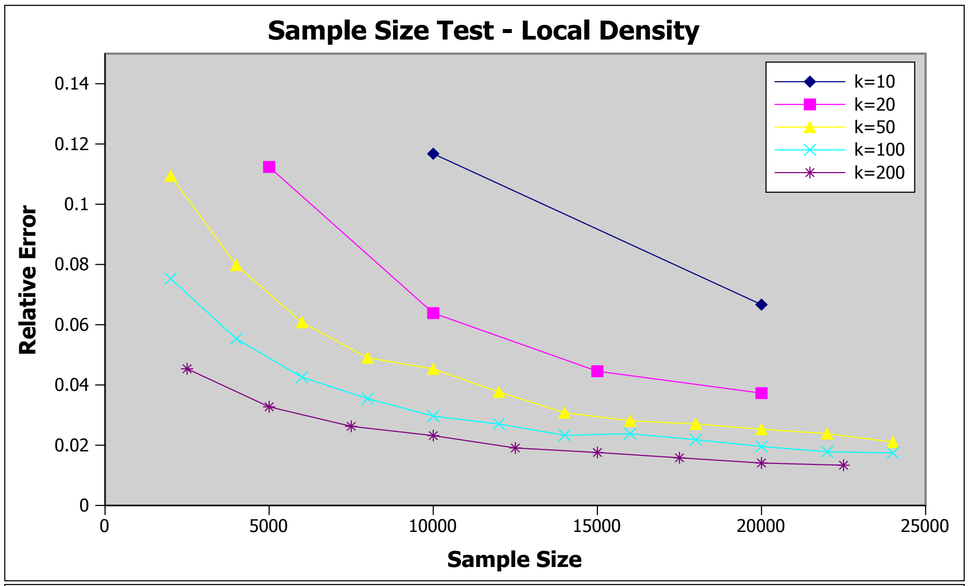


Figure 5.3: *Sample size test for the regression of the query range; for 'uniform vector' and 'corel' datasets.*



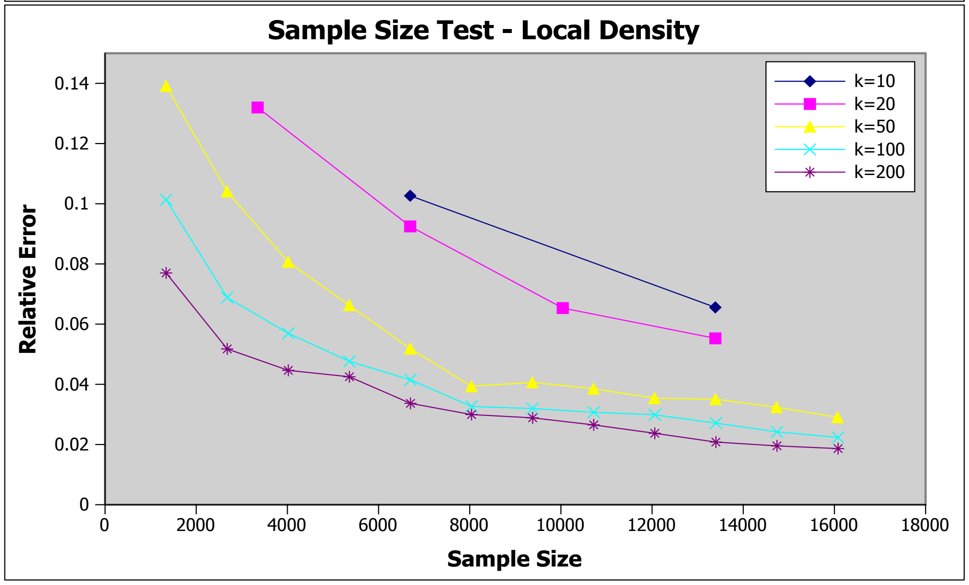


Figure 5.4: *Memory test for the local density method; for 'uniform vector' and 'corel' datasets.*

It can be observed that for small sizes of the sample set, the error increases. The reason is, the number of neighbors k' to be found becomes too low, decreasing the quality of the estimation. This can be related to the observation described earlier that the general distribution fails to successfully express the distance ratio for close objects. Also a conclusion can be made for the optimal size of the sample set, as its effect does not improve much after a certain value.

Projection of Distance Distribution

The shortcoming of this method comes through for small values of k', which is the number of neighbors to be retrieved for estimation. In cases where the size of the sample set can not be increased much, a new parameter k'' is used to replace k' when it's too small. Experiments for identifying an optimum value for k'' are presented in Figures 5.5-5.6 for different values of k.

An important observation from these results is that, greater values of k'' actually decreases the precision for *corel* data. The reason behind this is the global estimation used in proportioning in Equation 4.2 on Page 22. High values of k'' increases the affect of this ratio on the estimated k' radius, which also defines the actual k nearest neighbor distance. For uniform data the opposite is the case, where increasing k'' also increases the accuracy of the estimation, however in a slow rate for very high values.

The important point to be discussed in these experiments is that, increasing the sample set size also increases the computational overhead. The lower bound sorting approach is also applicable in this method since the only difference is to use $k'' > k'$ instead of small k' values. The required computations in memory for estimation is shown in Figures 5.7-5.8.

We decided to use 0.1 sampling rate for these methods, since for lower values, k' becomes less than 1 in which case the local density algorithm malfunctions. Using smaller sizes does not increase the error much, however, using the lower bound sorting technique, computational overhead can be minimized, and therefore higher number of sample objects can be efficiently processed.

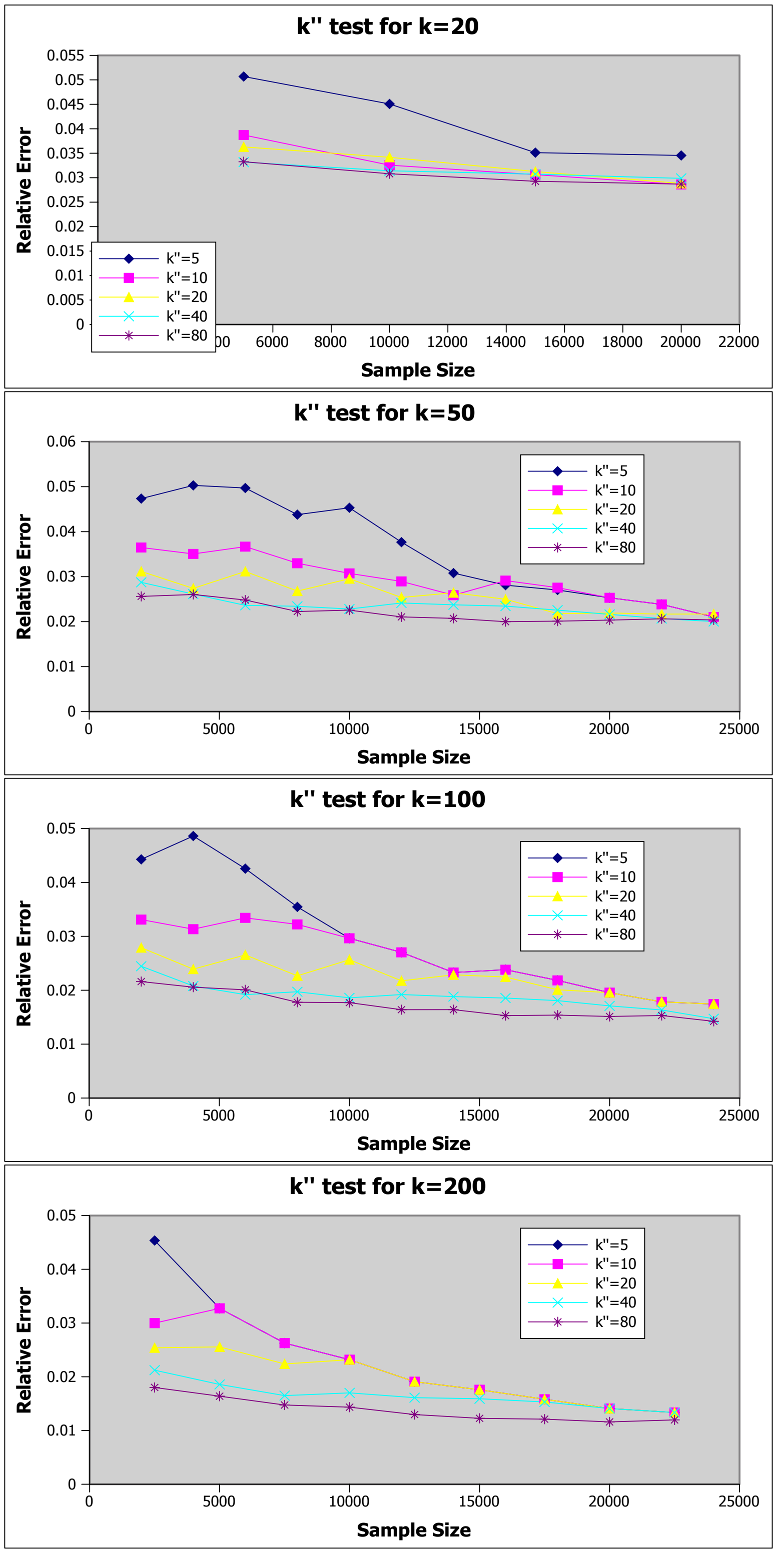


Figure 5.5: *Relative errors for different k'' values used in the local density method; k=20,50,100,200; for 'uniform vector' dataset.*

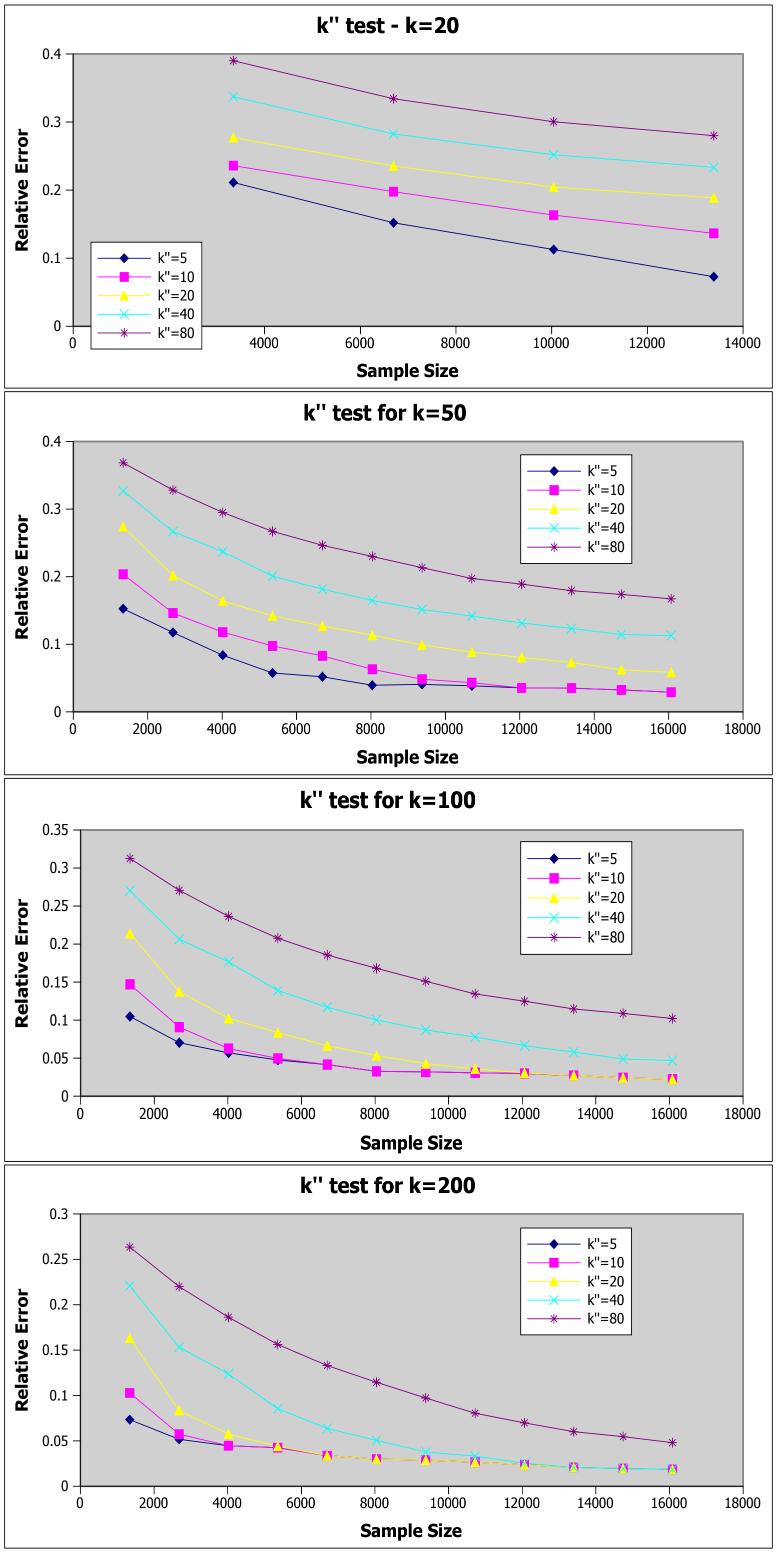


Figure 5.6: *Relative errors for different k'' values used in the local density method; k=20,50,100,200; for 'corel' dataset.*

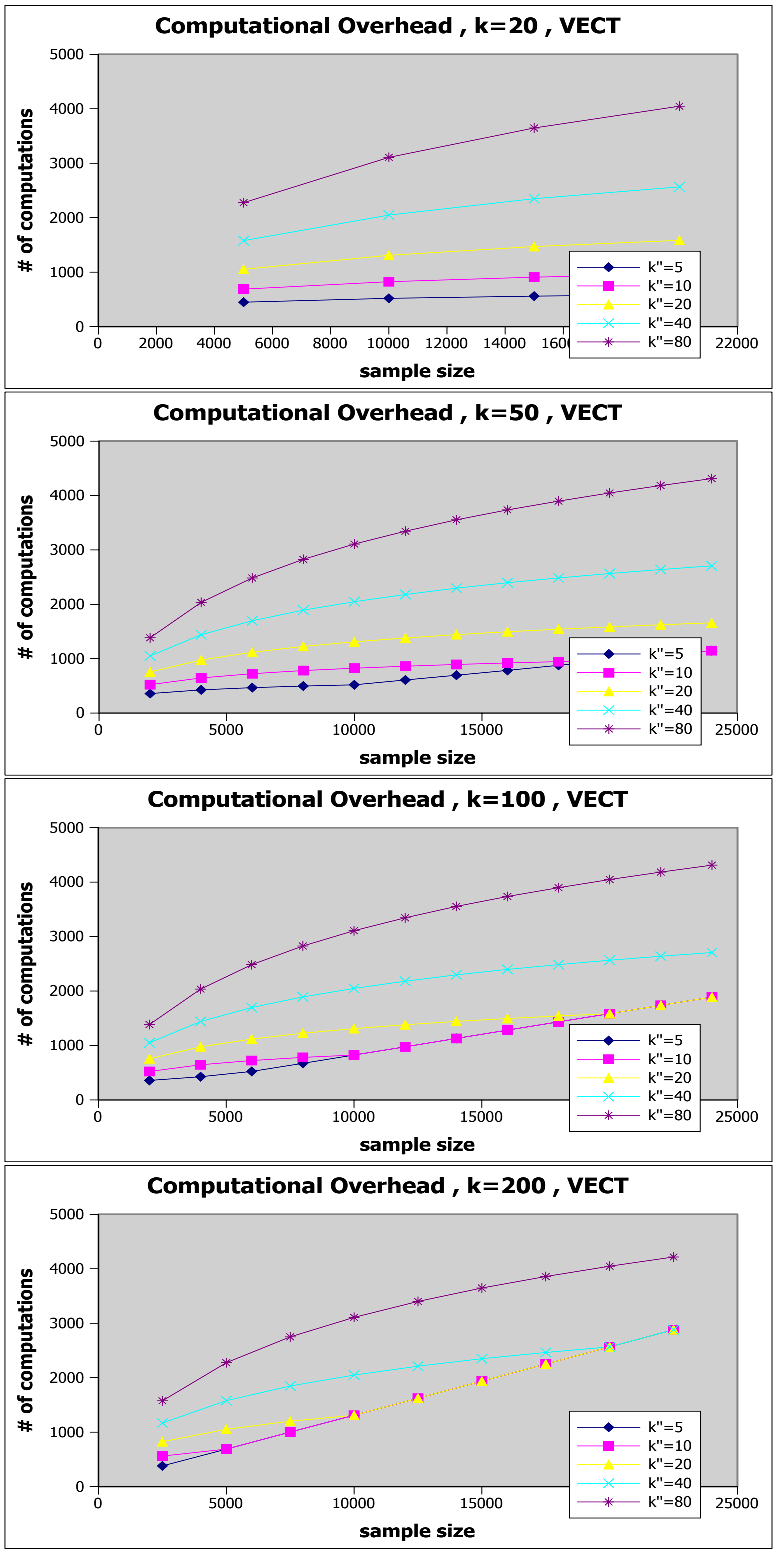


Figure 5.7: *Sample size test for the local density method; k''=20,50,100,200; for 'uniform vector' dataset.*

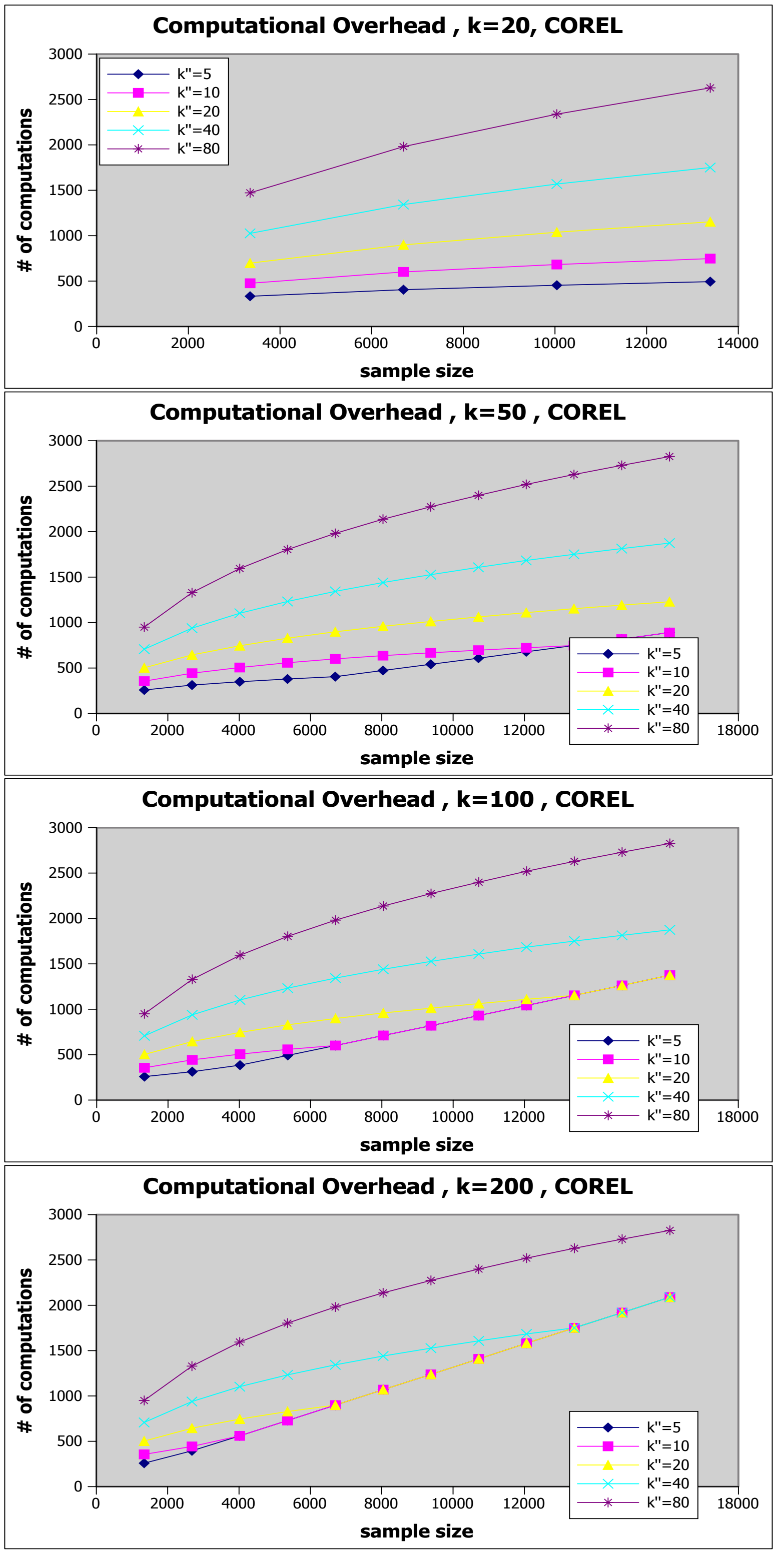


Figure 5.8: *Sample size test for the local density method; k″=20,50,100,200; for 'corel' dataset.*

The value of k'' is decided as 20, because larger values decrease performance for real datasets and only increase accuracy slightly for uniform data.

A key note is that for especially different sizes and dimensions these values can be altered for optimizing performances. Our choices are merely for fair comparison between different approaches.

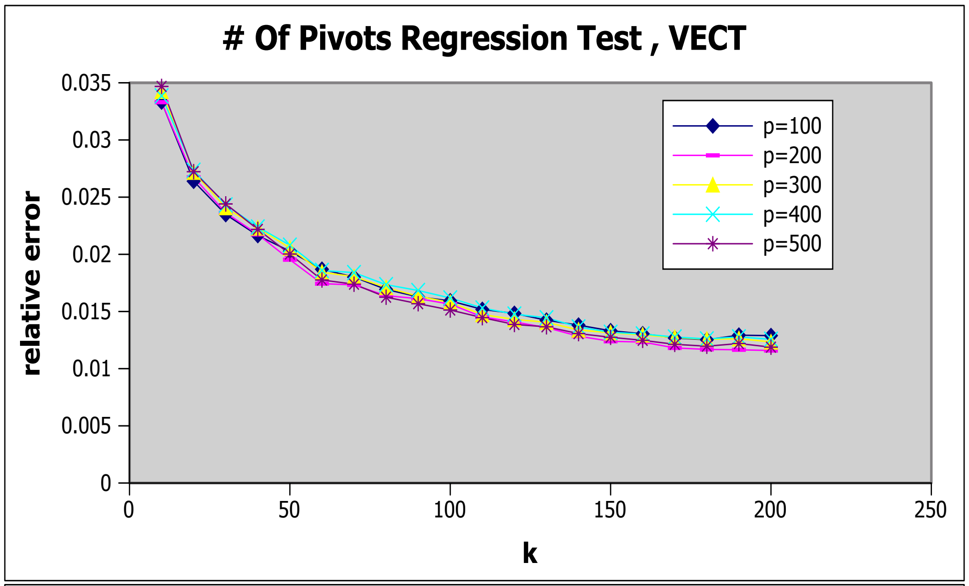
5.2.3 Static Pivots

Another approach to estimate the location of the query is by use of pivots. As mentioned before, a query is assumed distant to general population if its average distance to pivots is greater than expected. The effectiveness of the method based on the size of the pivot objects set is illustrated in Figure 5.9.

We see that the effect of changing the number of pivots differs according to the type of data. The increase in the number of pivots improves for corel data. However, we observe that after a certain value it has negative effect on accuracy of the estimation. This value changed for different experiments. In Figure 5.9, we see 500 pivots perform worse than using 300 pivots. On uniform data, on the other hand, the precision does not change considerably. This may be related to the low dimension size, since the precision is already very good.

Another important matter here is an increase in pivots size means higher computation during both construction time and radius estimation. For that reason, the positive effect of high number of pivots is insignificantly low compared to these costs, therefore 100 pivots were used during experiments. However, a better estimation can be made using 300 pivots, which may have significant effect on larger data sizes and dimensions, in exchange of a small increase in computational cost.

We will show the local estimations clearly perform better than global versions. The effect of the consideration of the query location is the main reason behind this. In methods that use the uniformity of local density, the CPU overhead is minimized by processing sample objects in sorted order of their lower bounds.



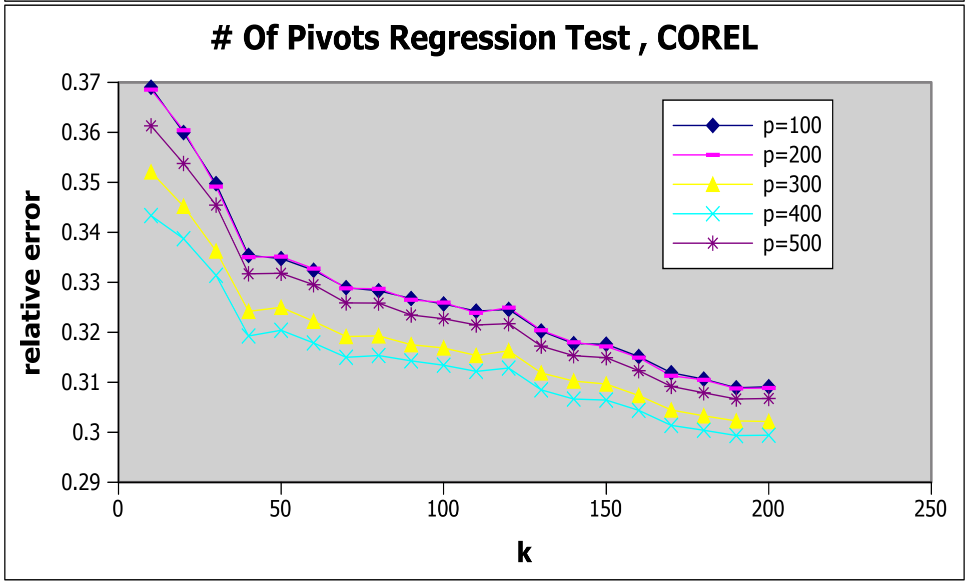


Figure 5.9: *Number of pivots test for training method; for 'uniform vector' and 'corel' datasets.*

However, the basic algorithm is used for regression of the query range, since the course of r is analyzed. This results in poorer performance in terms of number of computations. The training phase for static pivots does not require much additional computation, since pivot distances of every object are already pre-computed.

5.3 Overall Comparison

As we clarified, the performance of radius estimation can best be represented by its relative error over the actual k^{th} nearest neighbor distance. Besides that, the number of distance computations is compared along with the number of neighbors retrieved. An important relation between these two is that, when not enough neighbors are found, each algorithm continues to find the remaining neighbors using the lower bound sorting solution. This results in lower number of computations which may distort the performance measure by itself.

Lines in the graphics represents methods as follows:

GLOBAL	Global estimation using distance distribution
AVG-KNND	Global estimation using the average k^{th} nearest neighbor distance
LOCALD	Estimation considering the local density around the query
DDPROP	Same as LOCALD, except k” parameter is used for low k’ values
REG	Estimation from regression of <r distance> for objects processed in memory
FD	Estimation using the fractal dimensionality of the data
HIST1	Estimation using histogram of pivots k^{th} nearest neighbor distances
P-REG	Estimation from regression of <average pivot distance, k^{th} nearest neighbor distance>

The local estimation methods give better results compared to the global estimation approaches. In Figure 5.10, we see that the static pivots method using the information of average distances to pivots gives the best precision for estimation in uniform data. In difficult distributions, however, such as *corel* data, performance of methods using the information of local density gives better results relative to static pivots method. Local density method using k'' parameter estimates the radius of nearest neighbors very accurately for these datasets. In

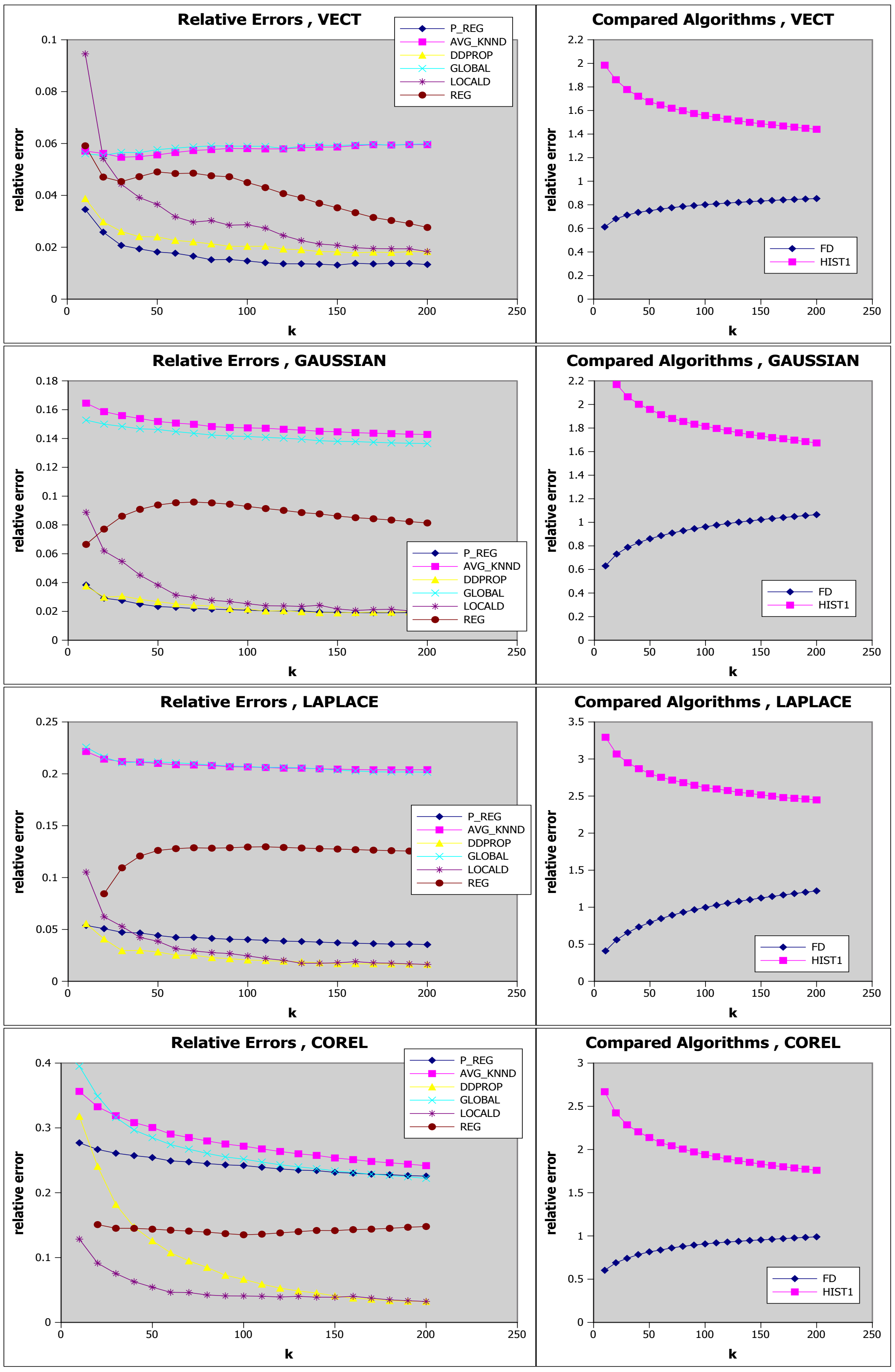


Figure 5.10: *Relative errors for different methods; for 'uniform', 'gaussian', 'laplace' distributions, 'corel' data.*

both cases the performance of algorithms using histograms and the fractal dimension fails to estimate the radius accurately. The increase in the relative error results from overestimation as the algorithms give significantly larger radius values compared to the actual k^{th} nearest neighbor distance. Although they return all k nearest neighbors, the number of distance computations rise excessively. A comparison of the estimation methods in terms of distance calculations is shown in Figure 5.11.

The methods using regression for the progress of the query range performs poorly, because of the inefficient processing of objects in the sample set. The reason is, as described, the objects are processed randomly, instead of in sorted order, so as to predict the course of the r value.

We observe that in terms of distance calculations, global estimations give similar results to local methods. This is actually insignificant, because the global estimation methods retrieve less number of neighbors, therefore requiring larger sizes of backup list. Since the candidate objects in this list is processed using the lower bound sorting approach, the total number of distance computations is kept in minimal values. The Figure 5.12 illustrates these observations.

We see that the less number of neighbors retrieved, the more objects the algorithm requires in the backup list. For some queries, global estimation retrieve even 0 neighbors therefore all the k-nearest neighbor are found from the backup list. In cases of higher dimensions and larger database sizes, this is unacceptable since the approach sorts the objects requiring additional disk scan and high CPU usage.

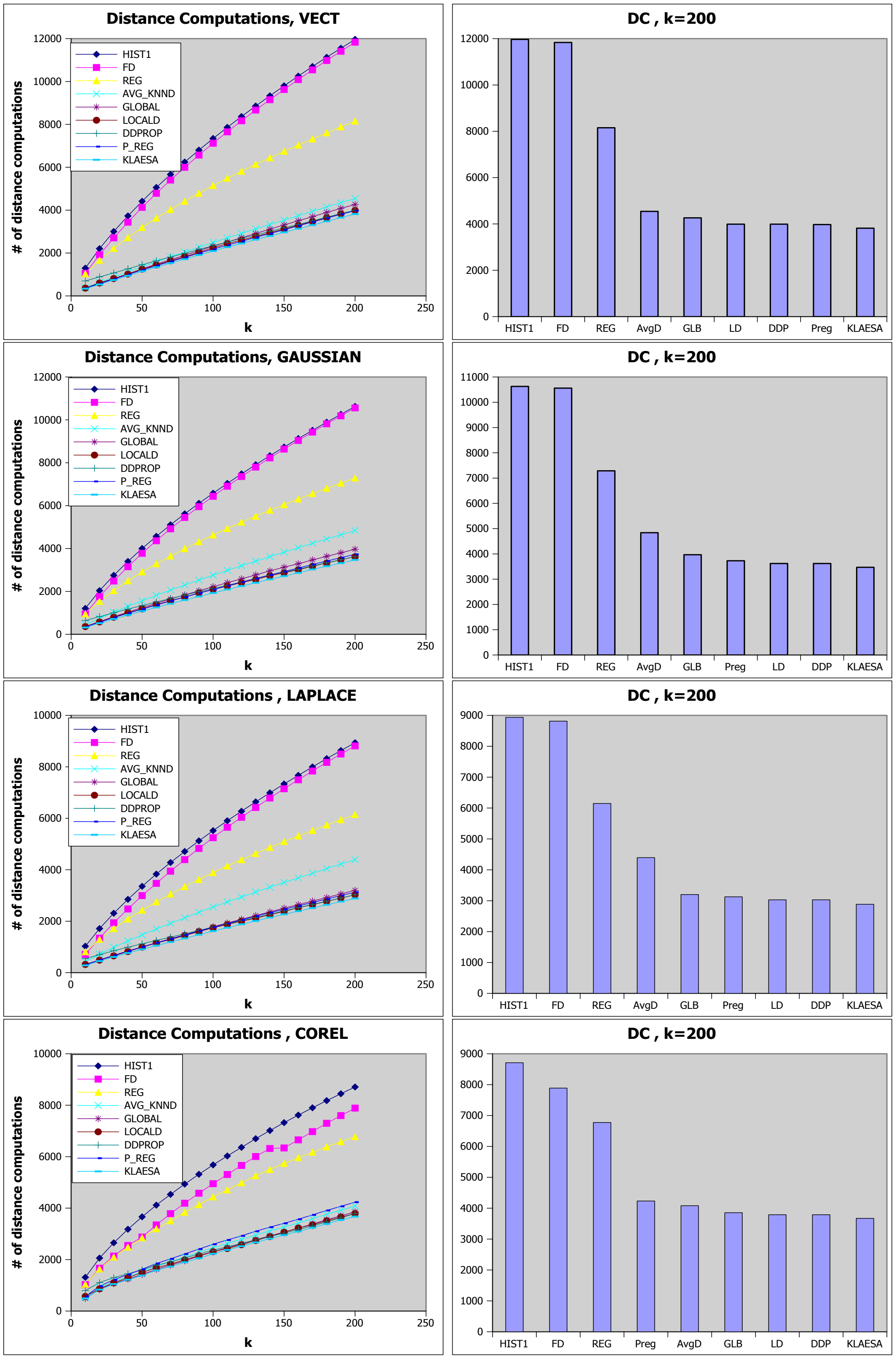


Figure 5.11: *Number of distance computations for different methods; for 'uniform', 'gaussian', 'laplace' distributions, 'corel' data.*

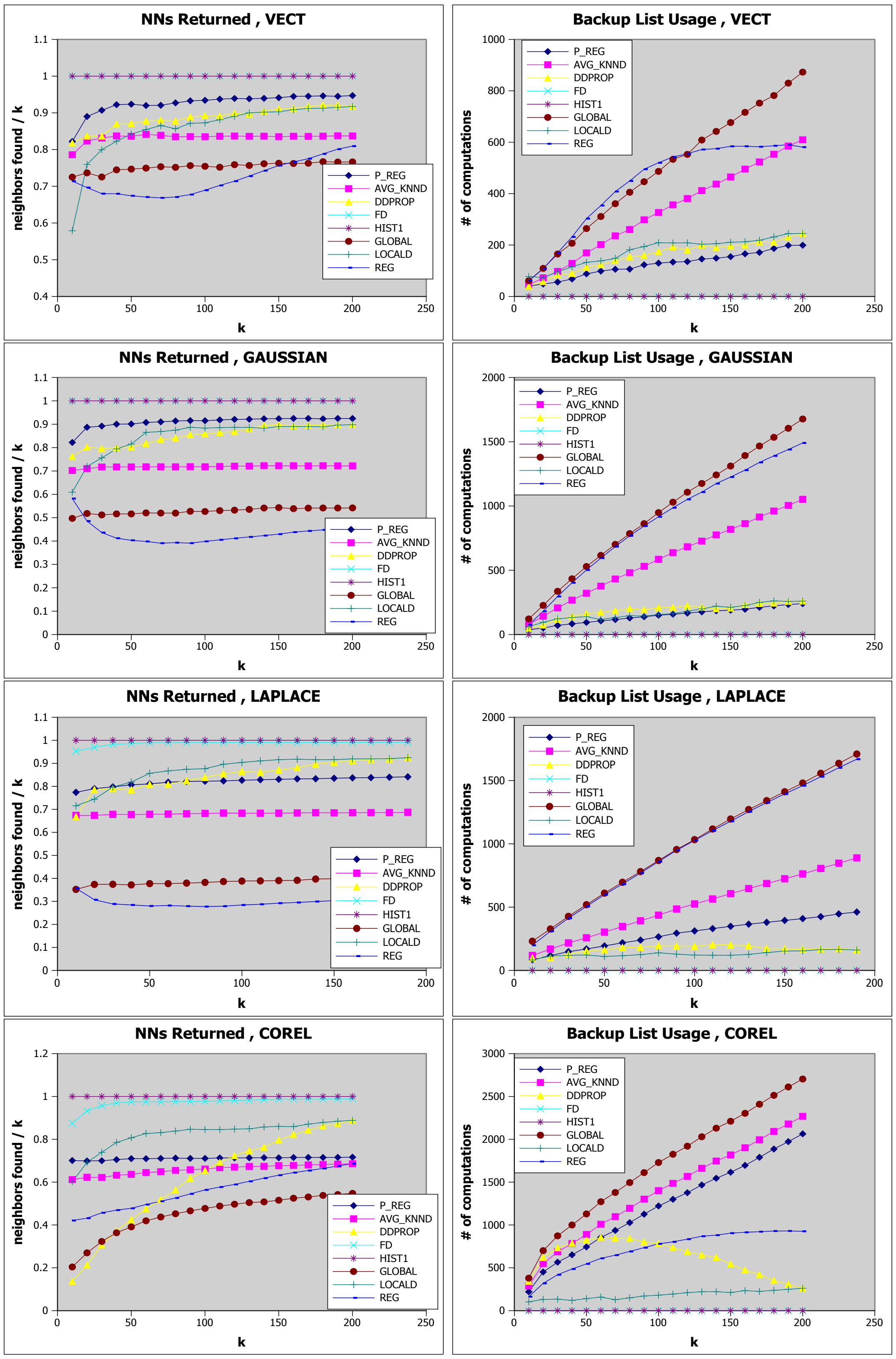


Figure 5.12: *Ratio of the number of returned neighbors and the size of the back-up list for different methods; for 'uniform', 'gaussian', 'laplace' distributions, 'corel' data.*

Chapter 6

Conclusion

In this thesis, we have presented an efficient k-nearest neighbor algorithm based on precise radius estimation. We proposed query processing using only one sequential scan of the index data, even not enough number of neighbors are retrieved by the estimation. A list storing the promising candidates for the remaining neighbors is shown to be kept at reasonable sizes by the accurate estimation of the k-nearest neighbor radius.

We have demonstrated the performance of different estimation methods emphasizing the amount of space and computation requirements. We illustrated their precision in terms of relative errors and the number of neighbors returned. We have shown that we outperform related algorithms using radius estimation without the requirement of significant computational overhead.

For uniform data, average pivot distances give precise information about the neighborhood of the query object, giving the best results compared to each other algorithm. On the other hand, the local density methods are shown to perform better compared to the static pivots method on difficult distributions. The reason behind this is, for non-uniform datasets, the distribution of pivots over the general population does not give precise information about the location of the query object. Since the density around a specific query is considered each time, other local estimation methods are not affected by difficult distributions that much.

6.1 Future Work

A number of improvements can be made to the proposed algorithm, especially regarding the local estimation methods. We have observed that the effect of the sample size changes for different values of k. Furthermore, it varies for different processing of the set. For regression of the query range method, for instance, the sampling rate can be decreased without losing much performance. This could decrease the computational overhead. The number of objects in the sample set can also be adjusted better for local density method when k'' is used as a parameter, since they have similar effects on the quality of the estimation.

Clustered data sets have different characteristics and we have tried different versions of the static pivots method. Instead of average values to all pivots, a single close pivot object can be considered, in order to obtain precise information about the cluster that the neighbors of the query object lies within. Similar ideas can also be adjusted for other local estimation methods.

Bibliography

[1] Sergey Brin. *Near neighbor search in large metric spaces.* In The VLDB Journal, pages 574584, 1995.

[2] Tolga Bozkaya and Meral Ozsoyoglu. *Distance-based indexing for high-dimensional metric spaces.* In SIGMOD 97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, pages 357368, New York, NY, USA, 1997. ACM Press.

[3] C. Celik. *Priority vantage points structures for similarity queries in metric spaces.* In Proceedings of EurAsia-ICT, volume 2510 of Lecture Notes in Computer Science, pages

[4] C. Celik. *Effective use of space for pivot-based metric indexing structures.* In: Proc.of Int. Workshop on Similarity Search and Applications (SISAP 2008), Cancn,Mxico, IEEE

[5] C. Celik *New Approaches to Similarity Searching in Metric Spaces.* PhD. Thesis.

[6] Edgar Chavez, Jose L. Marroqun, and Gonzalo Navarro. *Fixed queries array: A fast and economical data structure for proximity searching.* Multimedia Tools Appl., 14(2):113135, 2001.

[7] Edgar Chavez, Jose L. Marroqun, and Ricardo A. Baeza-Yates. *Spaghettis: An array based algorithm for similarity queries in metric spaces.* In SPIRE/CRIWG, pages 3846, 1999.

[8] Edgar Chavez, Gonzalo Navarro, Ricardo Baeza-Yates. *Searching in metric spaces.* ACM Comput. Surv., 33(3):273321, 2001.

[9] L. Chen and K.-H. Yap. *A fuzzy K-nearest-neighbor algorithm to blind image deconvolution.* IEEE International Conference on Systems, Man and Cybernetics, Vol. 3, pp. 5-8, Oct. 2003

[10] Paolo Ciaccia, Marco Patella, and Pavel Zezula. *M-tree: An efficient access method for similarity search in metric spaces.* In The VLDB Journal, pages 426435, 1997.

[11] C. Faloutsos, B. Seeger, A. J. M. Traina, and C. Traina Jr. *Spatial join selectivity using power laws.* In SIGMOD, 2000.

[12] Figueroa, K., Chvez, E. Navarro, G. and Paredes, R., *On the least cost for proximity searching in metric spaces.* WEA. 279-290, 2006.

[13] G. R. Hjaltason and H. Samet. *Index-driven similarity search in metric spaces.* TODS, 2003.

[14] L. Jin, N. Koudas, and C. Li. *Nnh: Improving performance of nearest-neighbor searches using histograms.* In EDBT, 2004.

[15] C. A. Lang and A. K. Singh. *Accelerating highdimensional nearest neighbor queries.* In SSDBM, 2002.

[16] Z. Lu and H. Burkhardt. *Fast Image Retrieval Based on Equal-average Equal-variance K-Nearest Neighbour Search.* In Proceedings of 18th Interna-tional Conference on Pattern Recognition, vol. 2.

[17] B. S. Manjunath and W. Y. Ma. *Texture features for browsing and retrieval of image data.* IEEE T-PAMI (Special issue on digital libraries) (Nov. 1996).

[18] L. Mico, 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 Recognition Letters, 15:917, 1994.

[19] F. Moreno-Seco, L. Mico and J. Oncina. *Extending LAESA fast nearest neighbour algorithm to find the k nearest neighbours.* Lecture Notes in Computer Science,vol. 2396, pp. 718-724, 2002.

[20] M. Schroeder. Fractals, Chaos. *Power Laws.* 1991.

[21] M. Tasan and Z. M. Ozsoyoglu. *Improvements in distance-based indexing.* In SSDBM, 2004.

[22] Caetano Traina Jr., Agma J. M. Traina, Bernhard Seeger, and Christos Faloutsos. *Slim-trees: High performance metric trees minimizing overlap between nodes.* In Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings, volume 1777 of Lecture Notes in Computer Science, pages 5165. Springer, 2000.

[23] C. Traina Jr., A. J. M. Traina, and C. Faloutsos. *Distance exponent: a new concept for selectivity estimation in metric trees.* In ICDE, 2000.

[24] Tosun U. *A New Dynamic And Adaptive Scheme For Indexing In Metric Spaces.* MSc. Thesis.

[25] Jeffrey K. Uhlmann. *Satisfying general proximity/similarity queries with metric trees.* Inf. Process. Lett., 40(4):175179, 1991.

[26] M.Vanco, G.Brunnett and Th.Schreiber. *A hashing strategy for efficient k-nearest neighbors computation.* Computer GraphicsInternational, 1999. Proceedings , 1999,Page(s): 120 128

[27] E. Vidal. *An algorithm for finding nearest neighbors in (approximately) constant average time.* Pattern Recognition Letters, 4:145, 1986.

[28] M.R. Vieira, C. Traina Jr., A.J.M. Traina, A.S. Arantes and C. Faloutsos. *Boosting k-Nearest Neighbor Queries Estimating Suitable Query Radii.* SS-DBM'07.