Similarity Search in Metric Spaces

Yuhui Wen

School of Computer Science

University of Waterloo

Waterloo, Ontario, Canada

N2L 3G1

Technical Report CS-2004-47

September 30, 2004

This report is a reprint of a thesis that embodies the results of research done in partial fulfillment of the requirements for the degree of Master of Mathematics in Computer Science at the University of Waterloo.

Abstract

Similarity search refers to any searching problem which retrieves objects from a set that are close to a given query object as reflected by some similarity criterion. It has a vast number of applications in many branches of computer science, from pattern recognition to textual and multimedia information retrieval.

In this thesis, we examine algorithms designed for similarity search over arbitrary metric spaces rather than restricting ourselves to vector spaces. The contributions in this paper include the following:

First, after defining pivot sharing and pivot localization, we prove probabilistically that pivot sharing level should be increased for scattered data while pivot localization level should be increased for clustered data. This conclusion is supported by extensive experiments. Moreover, we proposed two new algorithms, RLAESA and NGH-tree. RLAESA, using high pivot sharing level and low pivot localization level, outperforms the fastest algorithm in the same category, MVP-tree. NGH-tree is used as a framework to show the effect of increasing pivot sharing level on search efficiency. It provides a way to improve the search efficiency in almost all algorithms. The experiments with RLAESA and NGH-tree not only show their preformance, but also support the first conclusion we mentioned above.

Second, we analyzed the issue of disk I/O on similarity search and proposed a new algorithm SLAESA to improve the search efficiency by switching random I/O access to sequential I/O access.

Keywords - similarity search, metric space, pivot selction, pivot sharing , pivot localization, RLAESA

Acknowledgements

My greatest thanks are extended to my supervisor, Professor Frank Tompa, whose guidance and encouragement are the most important help to my study. He is a really friendly and responsible supervisor, who always gave me timely and valuable comments, who even kindly advised me during his holiday.

I would also like to give my special thanks to my first supervisor, Professor Gisli Hjaltason, who passed away during my study. His teaching and guidance gave me lots of help to understand the area of "Similarity Search" and triggered me to study in this area.

I want to thank my classmate, Lei Chen, for giving me important suggestions and helping me on some of the experiments.

Moreover, I would like to thank my readers, Professors Edward Chan and Alex Lopez-Ortiz, for their time in reviewing my thesis and their comments on improving my work.

Financial support from the University of Waterloo is gratefully acknowledged.

Contents

Chapter 1

Introduction

Similarity search is an important search problem in many branches of computer science, from pattern recognition to textual and multimedia retrieval. During the last decades, the applications of these branches have increased in many areas, such as geography, molecular biology, approximate text search, image and video retrieval, pattern classification and matching. The demand for a fast similarity search is thus increasing.

The similarity search problem is defined as follows: Given a set of objects and a query object, find the most "similar" object to the query object. Depending on the similarity criterion, multiple features from the objects or the whole contents of the objects are used to determine their similarity. By treating each feature from the objects as one dimension in a multiple dimensional space, the similarity search problem can be mapped into a nearest neighbour search problem in multiple dimensional spaces and the similarity criterion is then mapped into a distance function. Thus, similarity search is sometimes called nearest neighbour search and the results of similarity search are called nearest neighbours. In this thesis, these terms will be used interchangeably.

The definition of similarity search doesn't apply any restrictions on the number of dimensions. However, the problem becomes more difficult when the dimension increases. Since it is difficult to visualize a space with more than 3 dimensions, people tends to use 2 or 3 dimensional space as an analogy of higher dimensional spaces. However, we

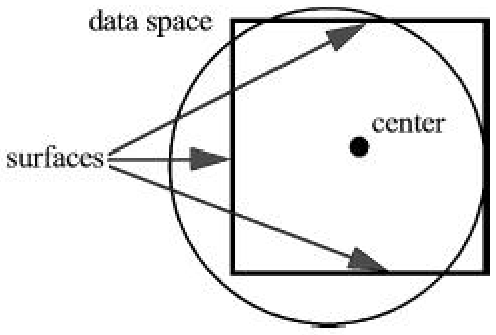


Figure 1.1: Sphere in high-dimensional spaces

must note that many properties in lower dimensional spaces do not remain in higher dimensions. To demonstrate this, let us consider the following examples, as presented by Bohm et al. [3]. In a cubic-shaped D-dimensional data space of extension $[0,1]^D$, the center point c of the data space is defined as $(0.5, 0.5, ..., 0.5)$. The lemma, "Every D-dimensional sphere touching (or intersecting) all $(D-1)$ dimensional surfaces of the data space also contains c", is obviously true for $D = 2$, as shown in Figure 1.1. It is not difficult to show that it is also true for $D = 3$. However, the lemma is definitely false for $D = 16$, as shown in the following. Define a sphere around the point $p = (0.3, 0.3, ..., 0.3)$. This point p has a Euclidean distance of $\sqrt{D \cdot 0.2^2} = 0.8$ from the center point. If we define the sphere around p with a radius of 0.7, the sphere will touch or intersect all 15-dimensional surfaces of the space but will not include the center point c.

One of the most important effects of high dimensions is that the volume grows exponentially with the number of dimensions. The volumes of the hypercube and hypersphere are the following:

$V(cube) = e^D$ where e is the length of the cube, and

$V(sphere) = \frac{\pi^{D/2} r^D}{\Gamma(1+D/2)}$ where r is the radius and $\Gamma(n)$ is the gamma function $\Gamma(n) = (n-1)!$

From the following ratio from the volumes of a cube and that of its embedding sphere, we will have some intuitive idea of "the curse of dimensionality". Let a sphere be embedded into a cube and touch each face of a d-dimensional cube. The radius r is half of the length e of the cube. Let the center of the sphere be the query object and retrieve all of the objects within the distance r. The ratio from the volume of the sphere to the volume of the cube is $ratio = \frac{V(sphere)}{V(cube)} = \frac{\pi^{D/2}}{\Gamma(1+D/2)\cdot 2^D} = \frac{1}{\Gamma(1+D/2)\cdot(4/\pi)^{D/2}}$. When the dimension D increases, the $ratio$ decreases at least exponentially. When $D = 3$, $ratio = \pi/6 \approx 0.524$; when $D = 10$, $ratio = 1/(5! \cdot (\frac{4}{\pi})^5) \approx 2.5 \times 10^{-3}$; when $D = 16$, $ratio = 1/(9! \cdot (\frac{4}{\pi})^8) \approx 4.0 \times 10^{-7}$. If objects are uniformally distributed in the cube, then the fraction of retrieval decreases exponentially. Therefore, if the traditional data structures are used to divide the data space into cubes, when the dimension is high, e.g. $D = 10$ or $D = 15$, most of the objects contained in any cube are uninteresting objects. Such divide-and-conquer searches are ineffective because they are like a sequential scan. In fact, any approach of using a set of cubes to contain the sphere will not work in high dimensions: If we fix the radius r of the sphere and the length e of the cube where $e < r$, when the dimension D increases, the number of cubes will increase exponentially. This means the space requirement of a data structure containing such cubes will increase exponentially and the number of non-interesting objects also increases exponentially. Moreover, it can be shown that using any fixed shapes instead of cubes will end up with similar results.

The "curse of dimensionality" can also be considered from another prospective. In high dimensional space, most of the volume is close to the surface of the data space, as is shown in the following example. Let us have a D-dimensional space with all the lengths equal to 1. In order to consider the region close to the surface, let us assume we only consider the locus of points with distance ≤ 0.05 from the surface. This defines a hollow cube with a volume $V = 1 - (1 - 2 \times 0.05)^D$. For $D = 3$, we have the volume $V = 1 - 0.9^3 = 0.27$. For $D = 10$, we have the volume $V = 1 - 0.9^{10} = 0.65$ and for $D = 15$, we have the volume $V = 1 - 0.9^{15} = 0.79$. Actually in any size and any shape of data space, the above property will be satisfied. Because of this property, if data objects are uniformally distributed in the space, progressively more of them will be close to the surfaces of the data spaces when the number of dimensions increases. Let the center of the cube be the query object, we can see the above property causes two effects on the nearest neighbour search. First, if the number of objects is fixed, the average distance of the nearest neighbour will be increased as the dimension increases. Second, the distances of all objects from the centre are becoming more and more similar when the number of dimensions increases.

These examples show that similarity search is a very hard problem and traditional approaches and thinking styles for lower dimensional spaces are not suitable for high dimensional spaces.

Currently the similarity search problems can be divided into two categories: vector space and metric space similarity search problems. In vector space similarity search, each object is represented as a point in D-dimensional space and its value on each dimension is available. Corresponding distance functions belong to the Minkowski L_r family: $L_r = (\sum_{1 \leq i \leq D} |x_i - y_i|^r)^{1/r}$. The best known special cases are $r = 1$ (Manhattan distance), $r = 2$ (Euclidean distance) and $r = \infty$ (maximum distance). The last distance deserves an explicit formula: $L_\infty = \max_{1 \leq i \leq D} |x_i - y_i|$, [18]. The data structures for vector spaces are designed to take full advantage of the precise location information. However, if such information is not available or not applicable, the data structures either can't be constructed or don't help the search. For example, if the object positions in each dimension are unknown, the data structures can't be constructed. It is quite different for similarity search in metric space. Even though each object is also treated as a point in some dimensional space, it is not mandatory to know the positions in each dimension, nor the dimensionality. It requires only a blackbox distance function, which given two objects returns a distance. In principle, one data structure can be used for any distance function. It can be seen that vector space similarity search is a special case of metric space similarity search. In this thesis, we are interested in the more general metric space similarity search instead of restricting ourselves to the vector space similarity search.

Distance functions in metric spaces have the following three common properties. The distance functions are defined as $d : U \times U \rightarrow R^+$ where U is the universe of the objects.

1. $d(x, y) = 0 \Leftrightarrow x = y$
2. $d(x, y) = d(y, x)$
3. $d(x, z) \leq d(x, y) + d(y, z)$

These three properities are valid for many reasonable similarity functions and definitely valid for the Minkowski L_r family. Note that since $d(x, y) \geq 0$, $\forall x, y$ can be derived from the above three properities, it is not listed as metric space properities as in other papers. As shown in Section 2, the third property, known as triangle inequality, is the key for all current metric space similarity searches.

In the previous examples, we have shown that the difficulity of the similarity search increases when the dimensionality of the data space increases. But the difficulty of the search also relies not only on the dimensionality of the data space, but also on the characteristics of the data objects [29, 5]. Let's consider the following well-known example [8]. We have a universe U of objects where $d(x, x) = 0, \forall x \in U$ and $d(x, y) = 1, \forall x, y \in U$ and $x \neq y$. To find a nearest neighbour, the query has to check all of the objects in the universe U. On the other hand, if there is a D-dimensional space and the universe U of objects in that space fall into one plane, the dimensionality of the search problem in this case should be 2 instead of D. To quantify the "actual" dimensionality and the search difficulity, the idea of intrinsic dimensionality was first introduced by Brin [5]. The basic idea is that intrinsic dimensionality grows when the mean of the distance is increased and the variance of the distance is reduced. Figure 1.2 gives some intuition why the search problem is harder when the variance is small. Let p be the data object and q be the query object. By the triangle inequality, object x can't have a distance to object q equal to or less than r if $|d(p, q) - d(p, x)| > r$ (the non-grayed

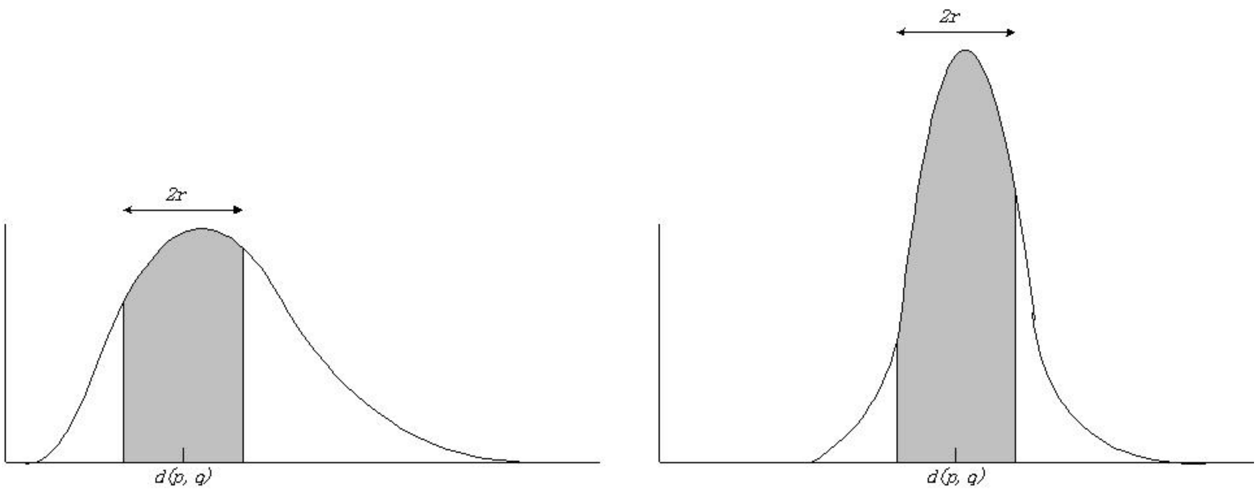


Figure 1.2: A flatter (left) versus a more concentrated (right) histogram. The latter implies the search will be harder because the triangle inequality discards fewer objects (the non-grayed area).

area). When the variance is small, more data objects will fall in the grayed area and few objects in the non-grayed area. Then the probability of discarding an object x will be lower. This phenomena is independent of the dimensionality of data space and only relies on the distribution of the data objects.

The definition of intrinsic dimensionality is given as follows:

Definition. The intrinsic dimensionality of a metric space is defined as $\rho = \mu_X^2/2\sigma_X^2$, where X is the random variable representing $d(p,q)$ for $p,q \in U$ and $\mu_X$and σ_X^2 are the mean and variance of X, respectively.

An interesting result is that the intrinsic dimensionality under the above definition matches the traditional dimensionality of the data space. As shown by Yianilos [30], the intrinsic dimensionality for a D-dimension space with uniformly distributed data objects and a distance function belonging to Minkowski L_r family is $\Theta(D)$ (although the constant depends on r).

Queries that are considered to be similarity search problems can be categorized into the following:

- Range Search: Given a query object q and a maximum search distance r, retrieve all of the objects from the universe U whose distances to q equal or are less than

r.

- k-Nearest Neighbour Query: Given a query object q and the number of requested objects k, retrieve the k objects in any order from the universe U whose distances to q are not larger than the distance of any remaining objects in U. This is usually represented as k-NN, where 1-NN is the special case in which only the nearest neighbour is returned.

- Incremental Nearest Neighbour Query: Given a query object q, retrieve the nearest neighbour o from the universe U and exclude o from consideration in the remaining steps of this query. Repeat this step until the query is stopped by the user or an external application. Such queries are referred to as INN, and both k-NN and INN together are referred to as NN search.

Range search is a simpler search than NN search, and many NN searches are built on range search. We will address this in detail in Chapter 2. The main difference between k-NN and INN are that the number of returning objects is given to k-NNs before the start of the query, while it is unknown to INN until the query is stopped. In principle, INN is a harder problem, and to avoid redundant calculations it needs to store many intermediate values for the query. As shown by Hjaltason [14], INN search can be implemented on the basis of k-NN search.

In many data retrieval applications, the similarity search is just a first step from which a large set of candidate objects are obtained and further filtering will be applied with more criteria before the final results are sent to the users. Many existing systems with this approach are listed by Baeza-Yates and Ribeiro-Neto [2]. For this kind of system, INN provides more flexibility than range query and k-NN queries because the external applications might not know the number of requested objects or the search distance in advance.

The total query time T is

$T = \#$ of distance computations $\times$ cost of $d()$ + extra CPU time + I/O time

In many applications, the cost of distance computations is much more expensive than the other two components. Thus, as shown by Chavez [8], many similarity searches evaluate the query time via the number of distance computations. We will apply the similar cost model in this paper, however, we also pay some attention to the I/O time in Chapter 5.

We have briefly introduced the characteristics of similarity search in this chapter. The remainder of this thesis is organized as follows. In Chapter 2, we cover the main previous work in metric space similarity search. In Chapters 3 and 4, we first analyze the advantages and disadvantages of various indexing structures currently used and propose that it is important to balance pivot sharing and localization when constructing the indexing structures. Two new algorithms, RLAESA and NGH-tree, are presented. RLAESA outperforms the state-of-the-art algorithm. NGH-tree is a generalized framework that can be applied to most of the current indexing structures. It shows how search efficiency can be improved by increasing the pivot sharing level and to what extent. The experiments with RLAESA and NGH-tree not only show their performance, but also support, with other experiments, the above conclusion about pivot sharing and localization. In Chapter 5, we addresse the issue of disk I/O and propose an approach, SLAESA, to reducing the disk I/O by using sequential instead of random access. Experiments show that SLAESA has shorter query time than the algorithm on which it is based. In Chapter 6, we summarize our contributions and list possible future work.

Chapter 2

Related Work

Many indexing structures have been proposed to facilitate similarity search in metric spaces. Most, if not all, of the indexing structures can be applied to range search and NN search. We will first explain how range search and NN search can be performed with these structures.

2.1 Range Search

Recall that range search is defined as the following: Given a query object q and a maximum search distance r, retrieve all of the objects from the universe U of the objects whose distances to q equal or are less than r. To facilitate the range search, many different indexing structures are used. In general, they can be divided into two categories, distance matrix structures and tree structures.

2.1.1 Distance Matrix Structures

2.1.1.1 AESA

One of the commonly used distance matrix structures is AESA (Approximating and Eliminating Search Algorithm), proposed by Vidal et al. in 1986 [24] and modified by Vidal et al. in 1988 [25] and in 1994 [26]. AESA performs surprisingly better than other algorithms by an order of magnitude, but suffers from its quadratic storage requirement [8]. According to Vidal, search time in AESA is constant and is not affected by the size of the data set. The distance matrix of AESA is constructed by the following method:

Let the universe of the data objects be U and its size be n. $\forall x, y \in U$, compute the distance $d(x, y)$. The distances are stored in an n by n matrix. Since the distances are symmetric, half of the elements in the matrix are redundant and can be removed. So, the space requirement of the matrix is $n(n-1)/2$ which is $O(n^2)$. It is clear that the complexity of distance precomputation is also $O(n^2)$. During a range search, an arbitrary object p is chosen as pivot and its distance to the query $d(p, q)$ is computed. By the triangle inequality mentioned in Chapter 1, all of the data objects $o \in U$ which don't satisfy $d(p, q) - r \leq d(p, o) \leq d(p, q) + r$ will certainly not be the results of this range search and can be eliminated from further consideration. The process of choosing an arbiratry object from the non-eliminated object set and eliminating more objects will be continued until the non-eliminated object set is empty. Since all of the eliminated objects are not the results of this range search and all of the other objects have already had computed distances, by comparing the computed distances with the range radius r, the result of this range search can be obtained. In 1994, instead of choosing pivots from the non-eliminating object set, Vidal proposed a heuristic function for pivot selection. Using this heuristic function, some speed can be gained in the search time. The heuristic function being used in this paper is the lower-bound distance function and denoted as $d_{lo}(q, o)$ for query q and data object o. It is defined in the followng:

The universe of the data objects U is composed of the set of computed objects, U_c and the set of non-computed objects, U_n, where U_c and U_n are disjoint. For query object q and data object o, $d_{lo}(q, o) = \max_{p \in U_c}\{|d(p, q) - d(p, o)|\}$.

The lower-bound distance function $d_{lo}(q, o)$ is the max difference value by appling the triangular inequality on all of the pivots obtained so far. It can be seen that the closer the object o is to the query q, the smaller the value $d_{lo}(q, o)$ is. It can also been seen that if the value $d_{lo}(q, o)$ is small, object o is likely close to the query q.

Experiments showed that the approach of choosing pivots by the heuristic function slightly out-performed the approach of choosing pivots randomly. By the triangular inequality, if $d_{lo}(q, o)$ is bigger than the range search radius r, we know that object o is not the result for this range search and can be eliminated from further consideration. Condition $d_{lo}(q, o) \leq r$ equals to condition $d(p, q) - r \leq d(p, o) \leq d(p, q) + r$. So, the lower bound distance $d_{lo}(q, o)$ is used both as a heuristic function for pivot choosing and as an object-eliminating condition in AESA.

The problems of the AESA approach are that the space requirement and construction time are quadratic in the size of the data set, which make this approach only suitable for very small data sets. Without these problems, AESA is a good approach in terms of search efficiency. Experiments show that AESA has constant search times with respect to the data size and significantly increasing search times with respect to the dimensionalities [24, 26].

2.1.1.2 LAESA

LAESA stands for Linear AESA and is a derivative of AESA proposed by Mico in 1994 [16]. The intension of LAESA is to smoothen the harsh requirements of space and construction time in AESA. Instead of computing and storing the pairwise distances of all of the data objects, it chooses a constant number of pivots, k of them, from the data set and computes and stores the distance from each of the k pivots to each of the data objects. Mico followed the suggestion [6, 20] that the pivots should be chosen as far away from the clusters as possible. But instead of finding the k furthest objects as pivots, Mico proposed a greedy algorithm to find the k pivots. The first pivot is chosen from the data set randomly. Compute the distances from this pivot to all of the objects in the data set and store these distances in the first row of a k by n matrix, where n is the size of the data set. These distances are also added into an accumulator array. The next pivot is chosen as that for which the (accumulated) distance is the largest. Keep repeating this, until all of the k pivots are chosen. Because it is based on largest accumulated distance, this greedy algorithm is sometimes called pivot selecion by largest accumulated distance. The space requirement and construction in LAESA is $O(k \cdot n)$ where k is a constant number and n is the size of the data set.

The range search in LAESA is a little bit different from in AESA. In LAESA, the order in which pivots are computed is random. However, all of the pivots must be computed to eliminate as many data objects as possible, before any non-pivot data object is computed. Because all of the pivots will be computed in each query, previous work [28] suggests that the order in which the pivots are computed can be arbitrary instead of random, and the order in which the pivots are applied to eliminate the data objects should be the ascending order of the distances to query. This will eliminate the data objects in the earlier stages of the qeury, which speeds up the search by reducing the extra CPU time, one of the three parts of the total search time.

Through experiments, it can be seen that the number of distance computations in a search is $k + O(n)$, when the dimensionality is fixed. The optimal choice for k grows as the dimensionality grows. When optimal k is too large, we set k to some large value such that both the space requirement and construction time are acceptable. As for AESA, the search time of LAESA increases significantly as the dimensionality increases.

2.1.1.3 Other Distance Matrices

Shapiro [20] proposed a search algorithm that is related to LAESA and uses a k by n matrix where k is the number of pivots and n is the size of the data set. The data objects are sorted by their distances to the first pivot p_1, and they are labelled as $o_{1,}o_2, o_3, \ldots.., o_n$. All distances from pivots to the query object are computed in the beginning of the search. The search is initiated in the object o_j where $|d(p_1, q) - d(p_1, o_j)| \leq |d(p_1, q) - d(p_1, o_i)|$ for any $i \neq j$. That is, object o_j is the object whose distance from pivot p_1 is most similar to $d(p_1, q)$. All pivots are used to compute the lower bound distance $d_{lo}(q, o_j) = \max_{\text{all pivot } p} \{|d(p, q) - d(p, o_j)|\}$. Compare lower bound distance with range search radius r and o_j can be eliminated if $d_{lo}(q, o_j) > r$. If o_j can not be eliminated, compute the distance $d(q, o_j)$ and output o_j as result if $d(q, o_j) \leq r$. Similarly, repeat the above steps on other objects in the order of $(o_{j-1}, o_{j+1}, o_{j-2}, o_{j+2}, ...)$. The search stops in either direction when the condition $|d(p_1, q) - d(p_1, o_i)| > r$ is satisfied. When the range search radius r is fixed, the number of distance computations in Shapiro's algorithm is the same as in LAESA, because they use the same eliminating conditions. But Shapiro's algorithm has a smaller CPU overhead due to the ordering of the objects by their distances to the first pivot, which does not require a linear scan on all of the objects as in LAESA. The efficiency for NN search will be discussed later.

Wang and Shasha proposed a dynamic programming approach [27]. During a search, two n by n matrices are used where n is the size of the data set. One matrix is used for lower bound distance $d_{lo}(q, o)$ and the other matrix is used for higher bound distance $d_{hi}(q, o)$. For any objects x and y, $d_{lo}(x, y) \leq d(x, y) \leq d_{hi}(x, y)$. Before the search, some of the distances between two random objects are precomputed and stored in the above two matrices and all other distances in the distance matrices are either 0 or ∞. Note that the distances along the diagonal are 0. During the search, the query object q is treated as the $n + 1$st object in the augmented distance matrices. Update the lower bound distance d_{lo} and upper bound distance d_{hi} so that they are as tight as possible whenever new distances are computed. Similar conditions are used to eliminate the objects. Like AESA, this algorithm has quadratic space requirement which makes it unsuitable for large data sets.

TLAESA is an algorithm combining the technologies from distance matrix and tree structures. We will discuss it in the subsection of tree structures below.

2.1.2 Tree Structures

2.1.2.1 VPT

VPT or vantage point tree is presented by Yianilos [29]. Independently, Uhlmann has proposed the same basic structure, which is called a metric tree [22, 23]. The root node of VPT represents the whole data set. A vantage point v is chosen randomly from the data set. Compute the distances from all data objects to the vantage points, find the median M, and store it into the node. Divide the data set into the left/inner subset S_l and the right/outer subset S_r such that $d(v, o) \leq M$ $\forall o \in S_l$ and $d(v, o) \geq M$ $\forall y \in S_r$. Note that $|S_l| = |S_r|$ or they have a difference of 1 . The left/inner subset S_l is represented by the left child of the root node and the right/outer subset S_r is represented by the right child of the root node. Choose another vantage point for the 2nd level of the tree. For each node in the 2nd level, divide the set of data objects it represents into left/inner and right/outer subsets and create the corresponding left and right child nodes. This is done recursively. One vantage point is selected for each level of the tree until the sizes of the subsets represented by the leaf nodes are smaller than some threshold.

Range searches start from the root of VPT. The left subtree will be visited if $d(v, q) \leq M + r$ and the right subtree will be visited if $d(v, q) \geq M - r$ where r is the range search radius. It is possible that both subtrees are visited. When leaf nodes are reached, compute all distances from the objects inside that leaf node.

VPT has a linear space requirement. It is argued that VPT has a range search complexity equal to the height of the tree $O(\log n)$ [29], but as pointed out, it is only true when the range search radius r is very small, so that both subtrees are rarely visited.

2.1.2.2 MVPT

VPT can be extended into MVPT (multi-vantage point tree) [4] by dividing each node into multiple subsets. Instead of being divided into two subsets by the median distance, each set in VPT can be divided into m subsets with equal sizes by $m-1$ cut-off distances. Moreover, for each object in the leaf nodes, the exact distances from that object to some number of vantage points are stored in the corresponding leaf nodes. These distances can provide further pruning before the distances of the objects are computed.

2.1.2.3 GHT

Generalized hyperplane tree, GHT, is proposed by Uhlmann [22]. A generalized hyperplane in a metric space is defined by two points p_1 and p_2, $p_1 \neq p_2$, and every point on the hyperplane has the same distances from both p_1 and p_2. Note that it is called "generalized" because it is a plane with Euclidean distance function and might not be so under other distance functions.

GHT is a binary tree being constructed recursively. Initially, two random points p_1 and p_2 are selected and a generalized hyperplane is defined accordingly. A space is divided into two parts, the data objects on the same side of p_1 belong to the left subtree and the data objects on the same side of p_2 belong to the right subtree. Repeat this to construct a binary tree recursively until the numbers of the objects in the leaf nodes are small enough. By the randomness of the algorithm, the expected height of GHT is $O(\log n)$.

Search starts from the root node and the distances from the query to the two points p_1 and p_2 are computed. For range search with query q and radius r, the search will visit the left subtree if $\frac{d(p_1,q)-d(p_2,q)}{2} \leq r$ and the right subtree if $\frac{d(p_2,q)-d(p_1,q)}{2} \leq r$. The reason these conditions are used is the following: If q is on the same side of p_2, we want to check if the objects on the p_1 side can be eliminated. Let o be an object on the same side of p_1. Object o can be pruned away if $d(p_1, q)-d(p_1, o) \geq r$ or $d(p_2, o)-d(p_2, q) \geq r$. Since o is on the same side of p_1, $d(p_1, o) < d(p_2, o)$, we have $d(p_1, q) - d(p_2, q) \geq 2r$. Again, it is possible to visit both subtrees. Repeat the above steps recursively. It is argued that GHT has better performance than VPT in higher dimensions [8].

2.1.2.4 GNAT

GNAT, geometric near-neighbour access tree, is an extended version of GHT. It was first presented by Brin [5]. Instead of choosing two points on each split, choose several points from the objects. With a number of points $p_1, p_2, ..., p_m$, the original spaces can be divided into m subspaces $S_1, S_2, ..., S_m$ such that $\forall o \in S_i$, $d(p_i, o) \leq d(p_j, o)$ and $i \neq j$. The subspaces are called Dirichlet domains, which are related to Voronoi diagrams except that they don't require Euclidean spaces.

To have better usage of the distance calculation, GNAT not only uses the pruning conditions in GHT, but also stores and uses the ranges of distances from each point to each Dirichlet domain. An m by m matrix is created for each node of GNAT, where its elements are pairs of distance with the format of $(\min_{i,j}, \max_{i,j})$ and $\forall o \in S_j$, $min_{i,j} \leq d(o, p_i) \leq max_{i,j}$. For a range search with query q and radius r, the jth subtree of a node can be pruned away if $\exists i$, such that $d(q, p_i)+r < min_{i,j}$ or $d(q, p_i)-r > max_{i,j}$. Note that the pruning conditions in GHT can also be used for further pruning.

GNAT takes $O(m^2 n)$ space and $O(nm \log_m n)$ construction time, where m is the number of points for each space division and n is the size of data set. Experiments [5] show that GNAT will beat VPT in search time only when the number of points, m, is bigger than 20.

2.1.2.5 SAT

Navarro [18] presented a search algorithm based on approaching the query object spatially, instead of the general divide-and-conquer methods. The data structure used in it is called a Spatial Approximation Tree, SAT. To construct the tree, a random object a is selected as the root from the data set. Initially, the set of a's neighbours $N(a)$ is empty. Compute all of the distances from objects to a and sort them by non-decreasing order. Now start adding objects into $N(a)$ by checking the objects in their non-decreasing distance order. If that object is closer to an object x in $N(a)$ than to a , that object will not be added into $N(a)$,but instead put into a "bag" associated with x. Otherwise, it will be added into $N(a)$. After checking all of the data objects, the set of neighbours $N(a)$ is obtained, each with a corresponding "bag" of objects. Object a is assigned to the root node. Each neighbour of a and the objects in that neighbour's "bag" are assigned to a subtree. Repeat the above procedures on each subtree recursively, except that the object assigned the root node of the subtree is no long selected randomly, instead it is the neighbour of a. The covering radius, $r(a) = \max d(a, x)$ where $x \in N(a) \cup \text{bag}(N(a))$, for each node is computed and stored as well.

The search starts from the root node a. If $d(q, b) - R(b) \leq r$ where $b \in N(a)$ and $R(b)$ is the covering radius of b, the subtree rooted at b can be pruned away. Here q and r is the query object and search radius of the range search, respectively. An additional pruning condition may be used to achieve further pruning [19]. Let object $c \in N(a)$ and c is the closet object to query q in $N(a)$. If $(d(q, b) - d(q, c))/2 > r$, the subtree rooted at b can be pruned away. The pruning condition is based on the same reasoning as in GHT.

The space requirement of SAT is $O(n)$ and it takes $O(n \log n / \log \log n)$ to construct the tree. The depth of SAT is $O(\frac{\log n}{\log \log n})$ on average.

2.1.2.6 MT

M-Tree, or, MT, was proposed by Ciaccia et al. [9]. The structure of MT has some resemblance with GNAT, since it is a tree structure where a set of representatives are chosen at each internal node. All of the data objects are stored in the leaf nodes. For each internal node (the root of some subtree), an object is chosen from the set of objects in its subtree and stored in that node as the representative of the whole set. Note that one object can be selected as representative more than once. Each representative t has a covering radius defined as $r_t = \max\{d(o, t)\}$, for all of the objects o it represents. Moreover, each representative, except the one for the root node, has a distance from itself to the representative for the parent node. This distance and the covering distance are stored in the same node where the corresponding representative is stored.

Range search in MT starts from the root node and traverses recursively down the tree. Two pruning conditions can be applied in MT. Let's consider a range query with query q and search radius r. Let an internal node have a representative t and its parent node have a representative s. If $|d(q, s) - d(s, t)| > r_t + r$ or $d(q, t) > r_t + r$, the subtree rooted at this node can be eliminated.

The main difference between MT and other tree structures is the way it handles insertion and deletion of data objects. During an insertion, the procedure traverses down the tree and repeatedly finds the best subtree to store the object. The best subtree is the subtree needing the smallest expansion on its covering in order to store the new object. In case of a tie, the subtree with its representative closest to the new object is the best subtree. On the leaf level, if the number of objects exceeds the storing capacity, the leaf node will be split into two leaf nodes. An entry will be inserted into the parent node to point to the new node. If the storing capacity of the parent node exceeds the limit, the parent node will be split recursively. The deletion is done by merging. The whole process is similar to that of a B-tree.

There are many criteria to select a representative and split a node. The best is to obtain a split which minimizes the sum of two covering radii.

MT is a balanced tree with linear space requirement. Experiments [9] show that it outperforms the vector space R-tree. MT is the only method so far that is efficient in terms of I/O cost as well as the number of distance computations.

2.1.2.7 TLAESA

Mico et al. [17] presented TLAESA, which can be treated as a combination of distance matrix and tree structures. The main difference between it and other tree-like structures is that it is able to break the dependency between the pivot selection and the object partition by the tree structure. In TLAESA, a certain number of pivots are selected as in LAESA. However, in addition to a distance matrix, a binary tree structure is used. Each node t represents a set of data objects S_t and stores a representative of that set, m_t , where $m_t \in S_t$. If t is leaf node, it contains exactly one object. Otherwise, $S_t = S_{t_l} \cup S_{t_r}$, where t_l and t_r are the left and right child, respectively.

TLAESA is built recursively. For the root node ρ, S_ρ is the whole data set and m_ρ is chosen randomly from the data set. The right child of a node t has $m_{t_r} = m_t$. The left child has $d(m_{t_l}, m_{t_r}) \geq \max(o, m_{t_r})\ \forall o \in S_t$ and $m_{t_l} \in S_t$. Moreover, $S_{t_r} = \{o \in S_t | d(o, m_{t_r}) \leq d(o, m_{t_l})\}$ and $S_{t_l} = S_t - S_{t_r}$. The recursion stops at the leaves which represent singleton sets.

For each node t in the tree, a covering radius $r_t = \max(d(o, m_t))$, $\forall o \in S_t$ is calculated and stored. Furthermore, the distances from every pivot to every representative of the tree are computed and stored into a distance matrix.

The search algorithm starts initially from the root node. For a range search with query object q and search radius r, the distances from the query to each pivot are computed. The representative of a tree node has a lower bound distance $d_{lo}(q, m_t) = \max\{|d(q,p) - d(p, m_t)|\}$, for all pivots p. By the triangle inequality, we can see that no object represented by node t will be within range r of the query q if $r + r_t < d(q, m_t)$. Thus, we can eliminate the subtree rooted at node t. The search traverses recursively down the tree. If reaching a leaf node, it computes the distance of the object in that node.

The space requirement of TLAESA is $O(n)$ where n is the size of the data set. Experiments [17] show that LAESA has better pruning ability while TLAESA has smaller time complexity. Thus, TLAESA is preferable when the data set is large or the distance computation is not very expensive. No experiment is done against other tree structures.

2.2 Nearest Neighbour Search

Most of the nearest neighbour searches are built over range searches, no matter which data structures are used. The only exception is presented by Clarkson [11], which is a GNAT-like data structure. Due to its lack of relevance to this paper, we do not discuss it here. The methods by which nearest neighbour searches are built over range searches can be grouped into the following three categories.

2.2.1 Increasing Radius

The k-NN search built on range search with increasing radius is defined as follows. Search query object q with fixed radius $r = a^i\varepsilon$ where $a > 1$ and ε is some small constant. Search starts with $i = 0$ and increases i repeatedly until the number of the returned objects reaches or exceeds k. Since the search cost increases exponentially in the search radius, the NN search cost here is close to that of the final range search with suitable search radius.

2.2.2 Decreasing Radius

The k-NN search can also be built over range search with decreasing radius. The search starts with search radius $r = \infty$. Whenever a query-object distance $d(q, o)$ is computed, it is compared with r . If $r > d(q, o)$, update $r = d(q, o)$. A list stores the k nearest objects returned by the range search so far. When a new object is returned, it is inserted into this list, then remove the object with the largest distance so that the list always contains k objects. The search stops when the range search stops. So, this NN search is a range search whose search radius r is narrowed down continuously.

Some examples of NN searches using this method are VPT [29] and GNAT [5].

2.2.3 Recursive Traversal with Priority

The search will be more efficient if the above k-NN search is able to decrease the search radius r in earlier stages of the search. Since this will lead to more objects being pruned in the search. A heuristic function is added to the original range search so that the subtree with higher probabililty to reduce search radius r will be chosen first.

For example, in SAT [19], a lower-bound distance is used for the heuristic function. A priority queue stores a set of possible subtrees and their associated lower-bound distances are used as keys. The smaller the lower-bound distance, the earlier the subtree will be chosen. After a subtree is chosen and removed from the priority queue, the distance from the pivot in the root of the subtree to the query is computed and search radius r is updated if necessary. Each direct child node of the root of the subtree represents a smaller subtree. All of these smaller subtrees are inserted back into the priority queue with their corresponding lower-bound distances. Repeat the above steps until the priority queue is empty.

Similar techniques are proposed by Hjaltason [13] and Uhlmann [22].

Chapter 3

Pivot Selection

Data structures to support similarity searches are used to eliminate objects from candidates as quickly as possible. In all indices, representative points, or pivots, are used for such purpose. The choice of pivots is thus critical to the performance of the index. Howerver, little is know about how to select a good pivot. Shapiro [20] recommends selecting pivots outside the clusters and Baeza-Yates [1] suggests using one pivot from each cluster. In this chapter, we will analyze probabilistically the eliminative powers of the pivots selected from different locations.

Consider a D-dimensional hypersphere with radius 1 and u data objects uniformly distributed inside it. We use the notation $V_D(r)$ to denote the volume of a D-dimensional hypersphere with radius $r \leq 1$. Let the pivot be selected at the center of the hypersphere. The distance from that pivot to any random data object is a random variable Z. Probability function $P(0 \leq Z \leq r) = V_D(r)/V_D(1)$ represents the percentage of data objects with distances in the range of 0 tos r. By definition, the density function is the derivative of the probability function, $f_Z(r) = \frac{dP(0 \leq Z \leq r)}{dZ} = D \cdot r^{D-1}$ (Figure 3.1).

Now let the pivot be an object extremely far away from the hypersphere data space, the distance from the pivot to any random object in the hypersphere is a random variable with the range of $[x-1,\ x+1]$ for some constant x. Let Y be a random variable in the range $[-1, 1]$ such that $Y + x$ is the distance from the pivot to an object

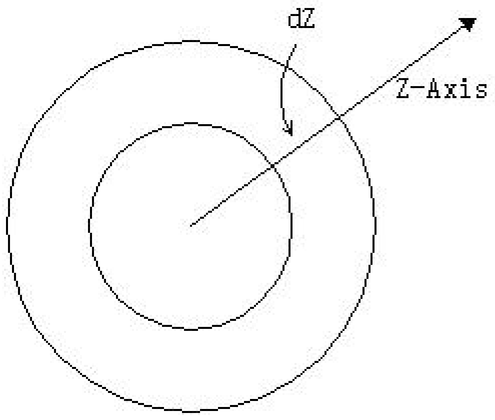


Figure 3.1: Pivot at the center

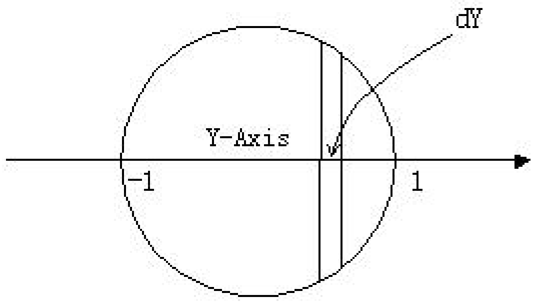


Figure 3.2: When the pivot is chosen exremely far from the hypersphere, the distance function can be viewed as the value on one coordinate.

3.2. Since the pivot is very far from the hypersphere, all objects that share the same value of Y can be considered to be coplanar. Thus, we have the probability function $P(-1 \leq Y \leq y) = \int_{-1}^{y} V_{D-1}((1-Y^2)^{\frac{D-1}{2}})dY/V_D(1)$ (see [15]).

The corresponding density function is $f_Y(y) = \frac{dP(-1 \leq Y \leq y)}{dY} = \frac{V_{D-1}(\sqrt{1-y^2})}{V_D(1)} = \frac{V_{D-1}(1) \cdot (1-y^2)^{\frac{D-1}{2}}}{V_D(1)}$.

The density functions f_Y and f_Z are shown in Figure 3.3. For any given query object q with $Y = y$ and $Z = r$, if $f_Y(y) < f_Z(r)$, the expected number of data objects near q with respect to the far pivot is less than the number with respect to the center pivot. Thus, the pivot far away has better pruning ability. Similarly, if $f_Y(y) > f_Z(r)$, the pivot at the center of hypersphere has better pruning ability. Since the hypersphere is symmetric with respect to the hyperplane that is perpendicular to Y and passing through the origin, for any query object q, we can always find another potential query object q' such that q and q' are symmetric to this hyperplane. If $Y = -y$ and $Z = r$

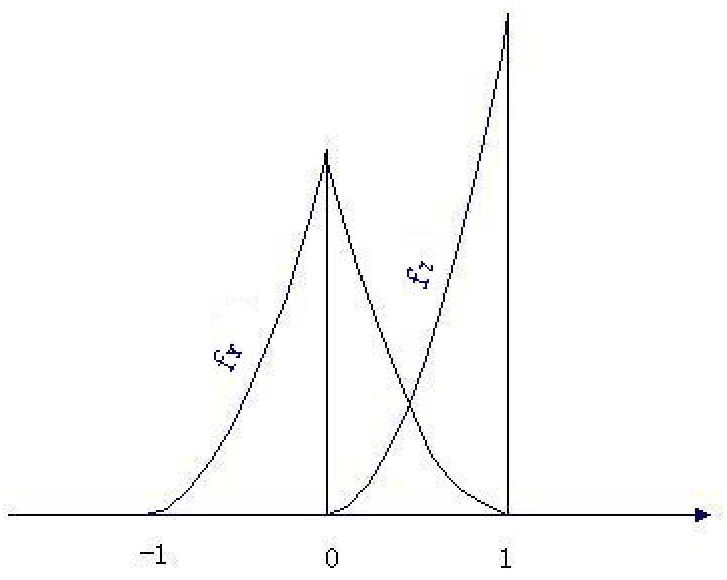


Figure 3.3: The density functions of Y and Z.

for query object q, we will have $Y = y$ and $Z = r$ for query object q'. Since the density function of Y is also symmetric $f_Y(-y) = f_Y(y)$, it is sufficient for us to consider the cases where $Y \geq 0$ only.

It can been seen from Figure 3.3 that the density functions f_Y for $Y \geq 0$ and f_Z have reverse trends. For any query object with $Y = y$ and $Z = r$, we want to discover if there exists a critical radius $r_c > 0$ such that when $r < r_c$, $f_Y(y) > f_Z(r)$ and when $r > r_c$, $f_Y(y) < f_Z(r)$. In a D-dimensional space, the location of query object q can be expressed as $(r, \theta_1, \theta_2, ..., \theta_{D-1})$ using a polar coordinate system. By the characteristic of a polar coordinate system, we know that $Y = r \cdot \prod_{i=1}^{D-1} \cos\theta_i$. Thus, $Y \in [-r, r]$. For the reason addressed in the previous paragraph, it is sufficient that we consider $Y \in [0, r]$ only.

When $Y = 0$, let $f_Z(r_c) = f_Y(0)$. Thus, we have

$$D \cdot {r_c}^{D-1} = \frac{V_{D-1}(1)}{V_D(1)} \cdot (1-0)^{\frac{D-1}{2}} = \frac{V_{D-1}(1)}{V_D(1)}$$

It can be shown that $\frac{1}{3} \cdot \sqrt{D} < \frac{V_{D-1}(1)}{V_D(1)} < \frac{1}{2} \cdot \sqrt{D}$. Figure 3.4 illustrates this property when the number of dimensions D is in the range of interest for any practical similrity search. Thus we have $D \cdot {r_c}^{D-1} = \beta \cdot \sqrt{D}$ where β is a number between $\frac{1}{3}$ and $\frac{1}{2}$. Thus, the solution follows:

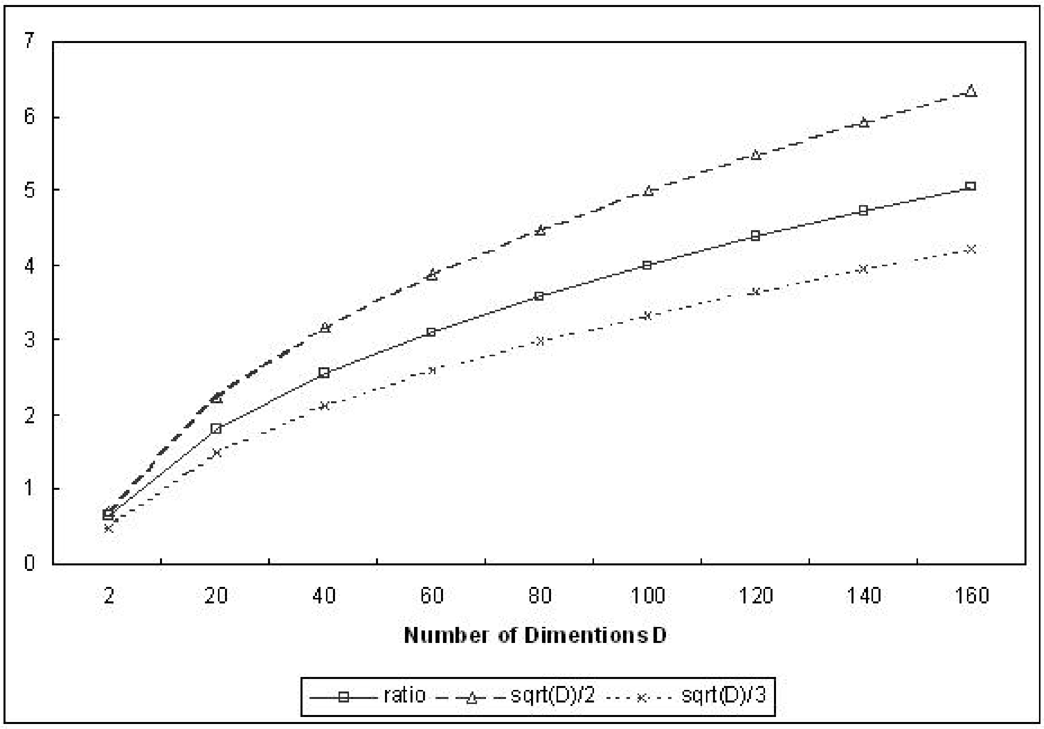


Figure 3.4: Bounding the ratio of $\frac{V_{D-1}(1)}{V_D(1)}$

By solving the equation, we have $r_c = (\frac{\beta}{\sqrt{D}})^{\frac{1}{D-1}}$.

When $Y = r$, we let $f_Z(r_c) = f_Y(r_c)$. Thus, we have

$D \cdot {r_c}^{D-1} = \frac{V_{D-1}(1)}{V_D(1)} \cdot (1 - {r_c}^2)^{\frac{D-1}{2}}$

$D \cdot {r_c}^{D-1} = \beta \cdot \sqrt{D} \cdot (1 - {r_c}^2)^{\frac{D-1}{2}}$

$(\frac{r_c}{\sqrt{1-{r_c}^2}})^{D-1} = \frac{\beta}{\sqrt{D}}$

Since $0 \leq r_c \leq 1$, r_c can be substituted by $\sin\alpha$ where $0 \leq \alpha \leq \frac{\pi}{2}$ and $(1 - {r_c}^2)$ by $\cos^2\alpha$. Then, we have $(\frac{\sin\alpha}{\cos\alpha})^{D-1} = \frac{\beta}{\sqrt{D}}$. Thus $\tan\alpha = (\frac{\beta}{\sqrt{D}})^{\frac{1}{D-1}}$, and we have $r_c = \sqrt{\sin^2\alpha} = \sqrt{\frac{1}{1+(1/\tan\alpha)^2}} = \sqrt{\frac{1}{1+(\frac{\sqrt{D}}{\beta})^{\frac{1}{D-1}}}}$.

So, for $0 \leq Y \leq r$, we have $\sqrt{\frac{1}{1+(\frac{\sqrt{D}}{\beta})^{\frac{1}{D-1}}}} \leq r_c \leq (\frac{\beta}{\sqrt{D}})^{\frac{1}{D-1}}$ and the critical radius r_c increases as the dimensionality D increases. Moreover, since $Y = r \cdot \prod_{i=1}^{D-1} \cos\theta_i$, Y is getting closer and closer to zero as the dimensionality D increases. Thus, r_c gets closer and closer to $(\frac{\beta}{\sqrt{D}})^{\frac{1}{D-1}}$ as the dimensionality D increases.

When the pivot is chosen inside the hypersphere but not at the center of the hy-

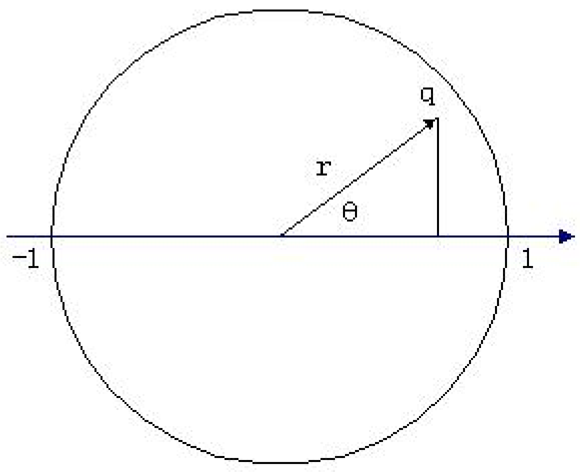


Figure 3.5: In a D-dimensional space, the query object q can be expressed as $(r, \theta_1, \theta_2, ..., \theta_{D-1})$in a polar coordinate system. By the characteristic of a polar coordinate system, we know that $Y = r \cdot \prod_{i=1}^{D-1} \cos\theta_i$

persphere, the critical radius r_c will be slightly different. The most extreme cases are shown in Figure 3.6. For the case demonstrated in the left graph, r_c will be smaller as the pivot moves away from the center. For the case demonstrated in the right graph, r_c will remain the same, no matter whether the pivot is away from the center or not. In a D-dimensional space, there is only one coordinate for each pivot satisfying the case in the left graph, while there are $D - 1$ orthogonal coordinates satisfiying the case in the right graph. As the dimension D increases, the effect caused by the case in the left graph becomes smaller and smaller. An alternative way to get this conclusion is as follows: Every pivot object p can be expressed in a polar coordinate system $(s, \theta_1, \theta_2, ..., \theta_{D-1})$ where s is the distance from the center of the hypersphere to the pivot. The longer the length of s projected on the Y axis is, the smaller r_c will be. But since the projection of s on the Y axis is $s \cdot \prod_{i=1}^{D-1} \cos\theta_i$, its length approaches zero as the dimensionality D increases. That is to say, when the dimensionality D is high, all of the pivots inside the hypersphere can be treated as if the pivot was at the center of the hypersphere.

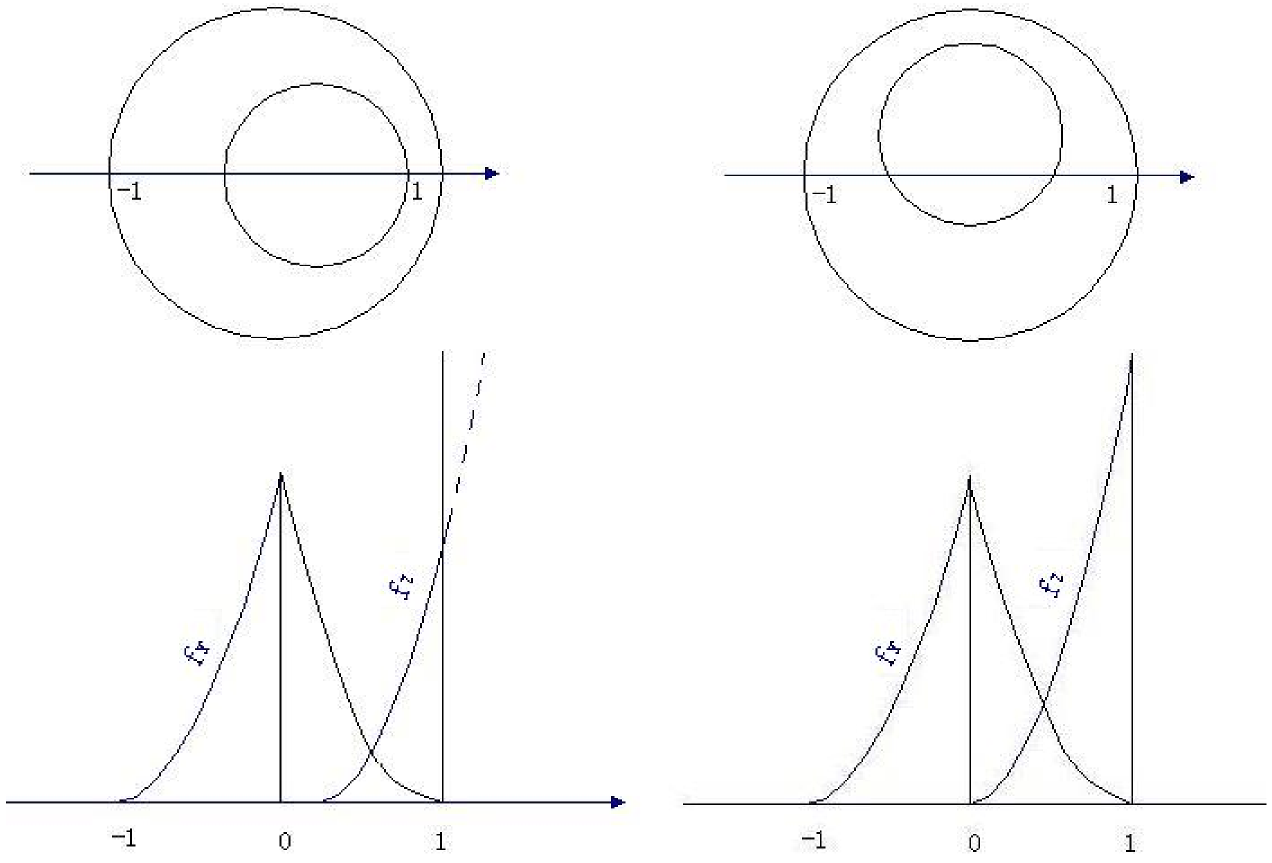


Figure 3.6: Both the graphs in the top show a pivot chosen inside the sphere while the other is chosen very far away. The graph in the upper left corner shows both pivots are on the same coordinate, while the graph in the upper right corner shows that the pivot from far away is on the coordiante and the other pivot is above the coordiante and has a projection on zero. The two graphs in the bottom show f_Y and f_Z for the corresponding cases above them.

By the density function, we have that

$E[Z] = \int_0^1 DZ^{D-1}ZdZ = \frac{D}{D+1} \cdot Z^{D+1} \, {}_0^1| = \frac{D}{D+1}$. That is, the expected distance from a random object to the pivot at the center of the hypersphere increases when D increases. In the previous paragraph, we have proven that the critical radius r_c is closer and closer to $(\frac{\beta}{\sqrt{D}})^{\frac{1}{D-1}}$ when the dimensionality D increases. With some calculation, we can prove that for dimension D and $D+1$, $\frac{E[Z_{D+1}]}{E[Z]_D]} < \frac{(\frac{\beta}{\sqrt{D+1}})^{\frac{1}{D}}}{(\frac{\beta}{\sqrt{D}})^{\frac{1}{D-1}}}$. The critical radius r_c has a bigger increasing rate than the average distance from the center.

Since $\frac{V_D(1)}{V_D(max(r_c))} = \frac{1}{(\frac{\beta}{\sqrt{D}})^{\frac{D}{D-1}}} = (\frac{\sqrt{D}}{\beta})^{\frac{D}{D-1}} > \frac{\sqrt{D}}{\beta}$, $\frac{V_D(1)}{V_D(r_c)}$ will increase at least proportionally to the square root of the dimension D. The fraction of objects inside the range of r_c to the pivot at the center will decrease as D increases. Thus, although we prefer to choose an object within r_c to use as a pivot, these objects are initially unknown and random selection of objects is unlikely to identify an object within r_c. Therefore, a pivot chosen within the hypersphere is less likely to have better pruning abilities than a pivot chosen from far away.

Chapter 4

Pivot Sharing And Localization

The techniques of pivot sharing and localization have been used widely in similarity search. However, no research has been done on the relationship between these two techniques. In this section, we will discuss their relationship in detail.

Pivot sharing level is used to indicate the number of pairwise distances from a certain pivot to other objects. The more distances available, the tighter the lower-bound and the more objects can be eliminated. Note that in some algorithms covering radius or range is used instead of pairwise distances. In this case, we treat the algorithm having the same pivot sharing level as if it has pairwise distances for each object inside the covering radius or ranges. Of course, now the pairwise distances are not the exact distances from the pivot to the objects. We will go back to this issue later in this section.

Pivot localization describes the distance relationship between a pivot and a set of objects. If the pivot is near all objects in the set, it is "local" to the set. Otherwise, it is "non-local" to the set. Pivot localization level is used to reflect the distance from the pivot to the set.

It can be seen that the pivot sharing level conflicts with pivot localization level. In this chapter, we will discuss how to use the balance between the pivot sharing and localization in order to reduce the query cost. Note that to make the comparison fair, the same storage spaces are used across different setting in the rest of this section if not mentioned explicitly.

4.1 Two Pivot Selection Techniques

Pivot selection techniques determine the balance of pivot sharing and localization. Note that pairwise distances are stored in non-tree structures while covering radii and/or ranges are stored in tree structures. Thus some papers, like Chavez [8], use the term "center" for tree like structures and "pivot" for non-tree structures. Here we still use the word "pivot" for both tree and non-tree structures. In general, the pivot selection techniques can be categorized into two kinds, divide-and-conquer and all-at-once.

Divide-and-conquer is seen in tree-structures only. It works as follows. A few pivots are selected first and the data set is divided into several subsets. The distances from objects of the current set to these pivots are computed and covering radii or ranges are stored. For each subset, a few other pivots are selected and the subset is divided into smaller subsets by space partitions. The above procedures are repeated recursively. No distances are computed and stored between an object and a pivot from a different subset. As the above recursive steps go deeper, the pivot localization levels increase since the pivots are chosen from smaller subsets via space partitions and pivot sharing levels decrease since only the distances from the pivots to the objects of the same subsets are computed and the sizes of the subsets decrease. Some algorithms that use this technique are GHT [22], GNAT [5], M-tree [9] and SAT [18].

The all-at-once technique selects all pivots from the whole data set in "one step". It is used in both tree-structures and non-tree structures. The pairwise distances from each object to each reference are precomputed and stored in non-tree structures while the ranges from each subset to each reference are used in tree structures. Pivot sharing level of a pivot reaches maximum since the distances from each object are precomputed. Pivot localization level is minimum because the pivot is chosen from the whole data set and no localization procedure is done. Some algorithms that use this technique are AESA [24], LAESA [16] and VPT [29].

Little is known about the performance of these two selection techniques. In practice, most algorithms choose pivots at random. Shapiro [20] recommends to select pivots outside the clusters, and Baeza-Yates [1] suggests using one pivot from each cluster. However, it is a common belief that the pivots should be far apart from each other, since close pivots give almost the same information.

4.2 Balancing pivot sharing and localization

In this subsection, we discuss how to balance pivot sharing and localization for different data sets. We show that for non-clustering data sets, higher sharing level and lower localization level should be used, while for clustering data sets, lower sharing level and higher localization level should be used.

4.2.1 Analysis

Theorem:

For a uniformly distributed data set in a hypersphere of radius 1, maximizing pivot sharing level and minimizing pivot localization level provide the best eliminating abilities.

Proof:

It is sufficient to show that all-at-once selection is superior to any divide-and-conquer selection.

Recall that in Section 3 we prove that the pivot should be chosen from as far as possible if we can not guarantee it is within distance r_c from the query. Moreover, we also prove that $\sqrt{\frac{1}{1+(\frac{\sqrt{D}}{\beta})^{\frac{1}{D-1}}}} \leq r_c \leq (\frac{\beta}{\sqrt{D}})^{\frac{1}{D-1}}$ where β is a number between $\frac{1}{2}$ and $\frac{1}{3}$, D is the number of dimensions, and the chance of choosing a pivot within distance r_c from the query in high dimensional spaces is extremely small. Thus the first pivot is always chosen from far away.

Now let's assume m pivots have been chosen from far away. We are going to choose pivot $m+1$.

If pivot $m+1$ can't be within distance r_c to the query, this pivot should be chosen from far away.

If pivot $m+1$ can be within distance r_c to the query, we need to consider whether to choose the pivot within distance r_c or from far away. Note that after obtaining the m distances from the first m pivots to the query, more information about the distances from the query to other objects are availalbe, such as lower-bound distances, even though the exact distances are unknown. If we used divide-and-conquer, we can capitalize on this information when choosing the next pivot. In this situation, assume in the best case we can eliminate all objects except those distances within r_c. Now we have a hypersphere of radius r_c with uniform distribution. Note that if we normalized the radius r_c to 1, we get back the same condition as if choosing the first pivot. Since the first pivot should be chosen from far away, so should pivot $m+1$.

So, all-at-once selection outperforms divide-and-conquer selection in this uniformly distibuted hypersphere.

It is a safe assumption that data sets with almost-uniform distributions and hypersphere-like shapes, such as hypercubes, should also have better query performance with all-at-once selection. That is, they should maximize the pivot sharing level and minimize the pivot localization level. Experimental results are shown later in this section.

For highly clustered data, lower sharing level and higher localization level should be used. That is, careful divide-and-conquer selection methods beat all-at-once selection for highly clustered data sets. The following examples are used for explanation. Let us assume there is a highly clustered data set which is composed of m clusters, whose inter-cluster distances are much larger than intra-cluster distances. For the first m pivots, we adopt Baeza-Yates's recommendation that we should choose one pivot from each cluster and each pivot is shared by all objects. For the next pivot, there must be

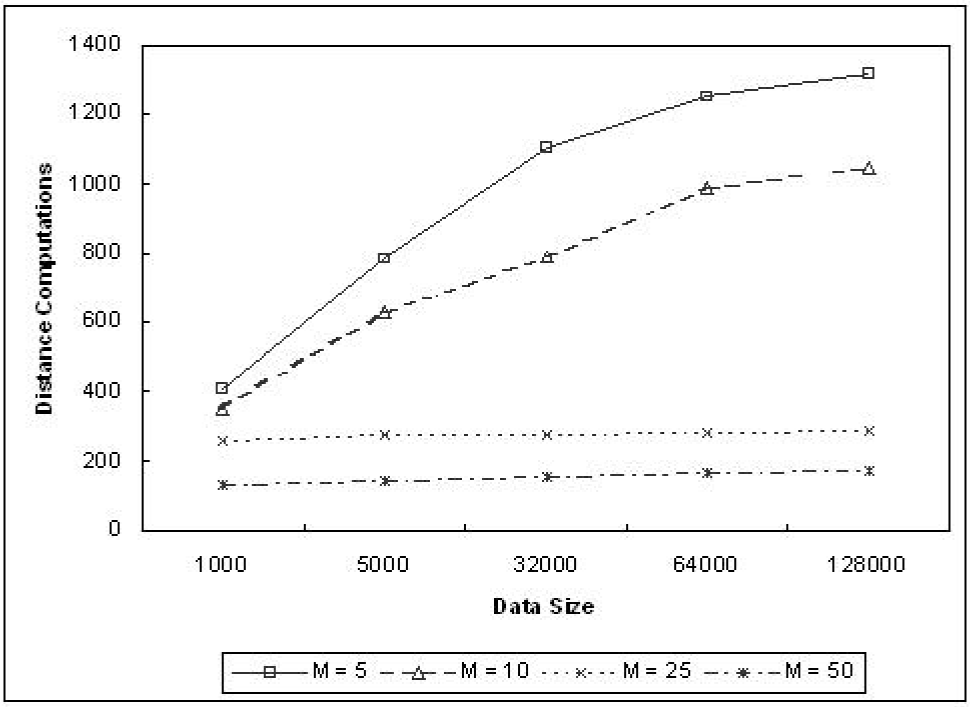


Figure 4.1: 10-dimensional data set with uniform distribution.

another pivot inside the same cluster with it. If pivot $m + 1$ is shared by all objects in the whole data set, it provides almost the same information to objects outside its cluster as the pivot in its same cluster. Thus, the next pivot should be shared by the objects inside the same cluster only. So, divide-and-conquer selection should be used in this case. For lower dimensional spaces, fewer than m pivots might be enough for eliminating the majority of the clusters. For data sets whose cross-cluster distances is not much larger than within-cluster distances, more pivots are needed before choosing locally. In general, a rule of thumb is to use lower sharing level and higher localization level for more highly clustered data.

4.2.2 Experimental Results

The following experiments show that all-at-once selection provides pivots with better eliminating power than divide-and-conquer selection when clustering is low. We chose GNAT [5], a divide-and-conquer algorithm based on hyperplane, to compare against

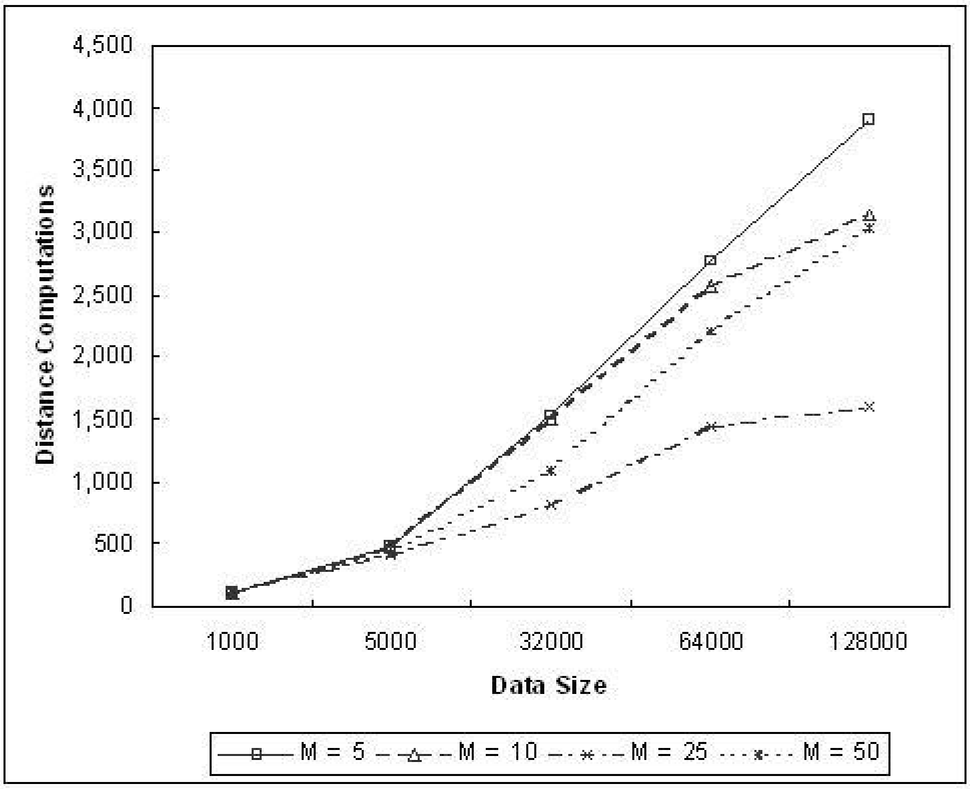


Figure 4.2: Effect of clustered data

LAESA [16], an example of all-at-once selection. To make the comparison fair, we modified GNAT so that pairwise distances from pivots to their objects are stored instead of ranges, thus allowing it to occupy as much storage space as does LAESA.

In the first experiment, data is uniformly distributed inside a hypercube with length 1 on all sides and dimensionality 10. LAESA uses 50 pivots for this situation. In modified GNAT, M pivots are chosen at each recursive step and there are $\frac{50}{M}$ recursive steps. Note that when $M = 50$, the algorithm is identical to LAESA. Experimental results are shown in Figure 4.1. It can be seen that the larger the value of M, the smaller the number of distance computations. LAESA has the best query performance here.

The second experiment is done on a clustered data set. The data set is created artificially as follows: A 15-dimensional hypercube with length 1 on each side serves as the data space. 10 objects are selected randomly from the hypercube, and these 10

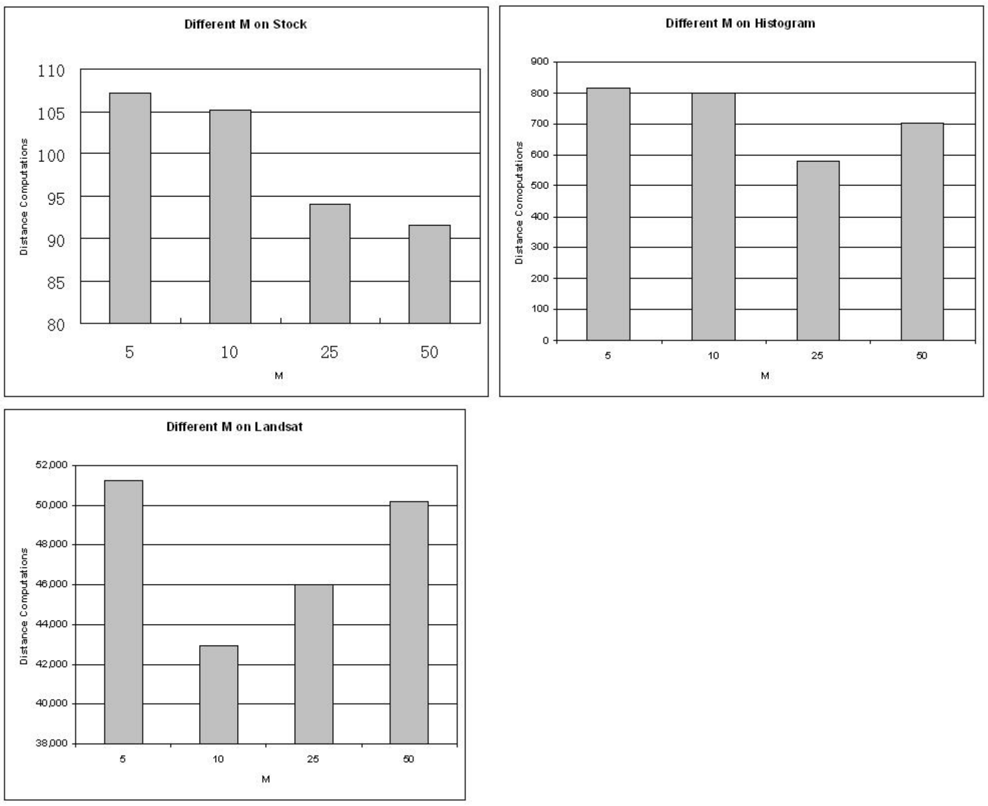


Figure 4.3: Effect of localization for real data sets

objects are used as cluster seeds. From each seed, more objects are created by altering each dimension of that seed with the addition of random values chosen from the interval $[-\varepsilon, \varepsilon]$, where ε is a small constant ($\varepsilon = 0.1$ here). The number of pivots M is defined in the same way as in the first experiment. Figure 4.2 shows that divide-and-conquer selection with $M = 25$ has the best query performance, gaining some benefit from localization.

Experiments are done on three real data sets: stock data, histogram data, and land satellite data sets [21]. These data have all been used as benchmarks for similarity search in multimedia databases [10, 12, 21]. The stock data set contains 6500 objects in 360 dimensions, and the first 30 dimensions are used for these experiments to avoid excessive computation time. Histogram contains 12,103 objects in 64 dimensions. The land satellite data set contains 275,465 objects in 60 dimensions. The stock data are scattered, whereas the histogram data and land satellite data are clustered. All experiments are repeated 5 times to get the average number of distance computations. The results are shown in Figure 4.3. 50 pivots are used in the experiment. That is, $M = 50$ again represents LAESA while the other are different setting on modified GNAT. It can seen that LAESA performs better on the scattered stock data, but localization works better on the two more clustered data sets.

These experiments verify our conclusion that all-at-once pivot selection has better query performance on scattered data sets, but divide-and-conquer pivot selection does better on clustered data sets.

4.3 RLAESA

RLAESA stands for representative LAESA and can be viewed as a extended version of LAESA. It is designed to take advantage of all-at-once selection while keeping the space requirement low. RLAESA is designed mainly for scattered data sets. However, thanks to its grouping methods, RLAESA outperforms previous algorithms using divide-and-conquer selection for all but highly clustered data sets.

4.3.1 LAESA

Since RLAESA can be viewed as an extension of LAESA, it is necessary to introduce LAESA in more detail [16]. It works as follows.

A set of k pivots are chosen so that they are almost as far away from each other as possible. The distances from each pivot to each of the n data object are computed and stored in a k by n matrix. This matrix and the k pivot indices are the only information stored in the indexing structures, taking $O(kn)$ space.

At query time, the distances from the query object to all of the k pivots are computed first. Put all these k pivots into a priority queue along with their actual distances as the keys. For all of the remaining objects, choose the pivot p having the smallest distance to the query and obtain the lower-bound distance $|d(p, q) - d(p, o)|$. Store these objects and their lower-bound distances into the priority queue. Pop up the object with the smallest distance from the priority queue. If the actual distance is computed, this object is the closest object to the query object. If the actual distance is not yet computed, apply the next pivot p_i to tighten the lower-bound distance. The new lower-bound distance is $\max\{|d(p_i, q) - d(p_i, o)|, \text{ prev lower bound distance}\}$. If all of the pivots have been applied to tighten the lower-bound distance, compute the actual distnace to the object. Put this object back into the priority queue with its new distance. Pop the next object from the priority queue, and repeat the above procedure.

The query time performance in LAESA is surprisingly good if the optimal k is used. It is reported that experiments show that it uses $O(1)+k$ distance computations [16, 8]. However, LAESA requires $O(kn)$ space and optimal k is usually very large and grows exponentially in the number of the dimensions. This makes LAESA less suitable for high dimensional spaces and/or large data sets, as compared to the tree structures. In the remainder of this chapter, we propose RLAESA to alleviate the problem.

4.3.2 Construction Process for RLAESA

We pick k pivots from the data set, as in LAESA. Next, we cluster the data set of size n into g groups where $g = n/k$. Each group is formed by first choosing a representative object at random from the object set. Other objects are assigned to the groups as described below. For each pivot and each group, the lower range is the smallest distance from that pivot to any object in that group while the upper range is the largest distance. Store the lower range from each pivot to each group in a two dimensional array, called *lMatrix*; similarly, we use *uMatrix* for upper ranges. For each group, compute the distances from each object in that group to the representative of the same group and store them in a two dimensional array, gDistances.

Now, let's go back to the grouping methods. There are three grouping methods proposed.

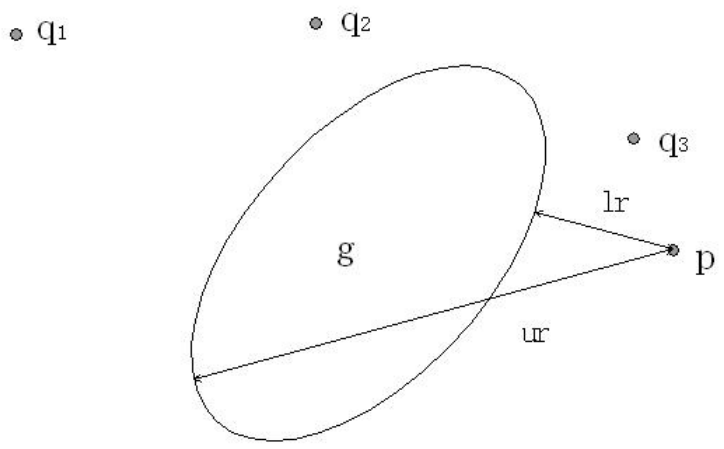


Figure 4.4: Lower-bound distances for group

- Method 1: Each object in a group is no further from the representative of that group than from any other representatives. That is, each object o in a group has $d(o, r) \leq d(o, r')$ where r is the representative of the group and r' is any other representative.
- Method 2: Each object o in a group has $\max_{p \in P} |d(p, r) - d(p, o)| \leq \max_{p \in P} |d(p, r') - d(p, o)|$ where r is the representative of that group, r' is any other representative and P is the pivot set.
- Method 3: Each object o in a group has $\sum_{p \in P} |d(p, r) - d(p, o)| \leq \sum_{p \in P} |d(p, r') - d(p, o)|$ where r is the representative of that group, r' is any other representative and P is the pivot set

4.3.3 Nearest Neighbour Search with RLAESA

Assume that given a query object and a data set, we want to find the nearest neighbour for that query. First, all of the actual distances from the pivots to the query are computed. The lower and upper ranges are then used to determine lower-bound distances for all objects in each group. Figure 4.4 gives an intuitive explanation, where for pivot p and group j, lr and ur are the lower range stored in $lMatrix$ and the upper range stored in $uMatrix$, respectively. In can be seen that with query object q_1, q_2 and q_3, the lower-bound distance for this group is $d(p, q_1) - ur$, 0 and $lr - d(p, q_3)$, respectively. If a group cannot be eliminated by its lower-bound distance, we compute the actual distance from its representative to the query object. Since the distances from that representative to all of the objects in the group are pre-computed, they can be used to tighten the lower-bound distance to each object. The lower-bound distance of each object is the maximum of the group lower-bound distance and the lower-bound distance via the representative. Thus, some more individual objects can be eliminated. If an object still cannot be eliminated, we compute its actual distance from the query object.

The details of the algorithm follow:

1. Compute all of the distances from the pivots $p_1, p_2, ..., p_k$ to the query.

2. Construct a priority queue which is minimum-oriented (that is, smaller keys have higher priority).

3. For each group j, apply the lower and upper ranges with the pivot p_1 to obtain its lower-bound distances ld using $ld = \max\{lMatrix[1][j] - d(p_1, q), d(p_1, q) - uMatrix[1][j], 0\}$. Put these groups into the priority queue with their lower-bound distances as keys.

4. Repeatedly retrieve the element with the smallest distance from the priority queue.

   (a) If the element is a group, check if more pivots can be applied to tighten its lower-bound distance.

      i. If yes, use the next available pivot p_i to tighten the lower-bound distance of this group j. The new lower-bound distance is $ld = \max\{lMatrix[i][j] - d(p_i, q), d(p_i, q) - uMatrix[i][j], ld\}$. Put this group back into the priority queue with the new ld.

ii. If no, compute the distance $d(r, q)$ from its representative to the query. Use this distance to tighten the lower-bound distances of each object in the group. Let the lower-bound distance of this group j be ld. The lower-bound distance for each object o is obtained by $\max\{|d(r,q) - gDistances_j[o]|,\ ld\}$. Put each object, other than the representative, and its corresponding lower-bound distance into the priority queue. Since the representative-query distance is computed, it is put back into the priority queue with the actual distance instead of the lower-bound distance.

(b) If the element is an object with its lower-bound distance, compute its actual distance to the query object. Put this object and its actual distance back into the priority queue.

(c) If the element is an object with its actual distance, return the object and stop.

4.3.4 Storage Space and Construction Cost Analysis for RLAESA

Two $k \times g$ matrices, $lMatrix$ and $uMatrix$ are needed to store the lower and upper ranges using $O(n)$ space, since $g = n/k$. Each group has an array $gDistances$. Distances from each object in the groups to the representative of the same group are stored. This takes $O(n)$ space, since each object has one distance stored. Pivots and representatives also take $O(n)$ space. So, the space requirement for RLAESA is linear in the data size.

The number of distance computations during construction is composed of that for the grouping procedure and that for preparing lower and upper range matrices. For methods 1, 2 and 3, it can be shown that the numbers of distance computations in the grouping procedure are $O(gn)$, $O(kn)$, and $O(kn)$, respectively. To construct the lower and upper range matrices, the actual distance from each object of a group to each pivot needs to be computed, which takes $O(kn)$ distance computations. Thus, the numbers of distance computations are $O(gn + kn)$, $O(kn)$ and $O(kn)$ for methods 1, 2 and 3,

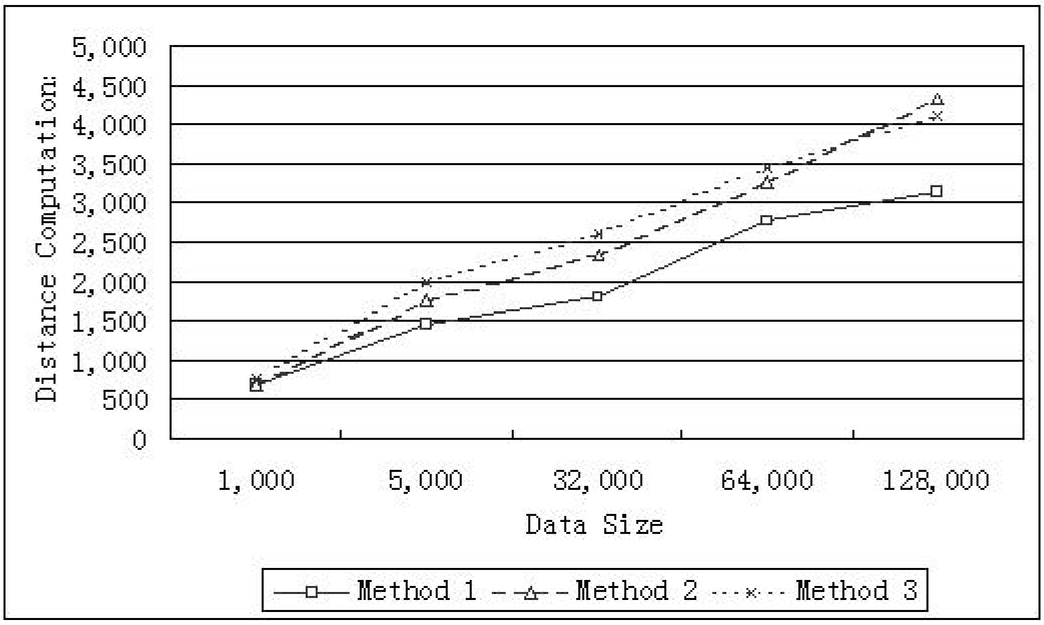


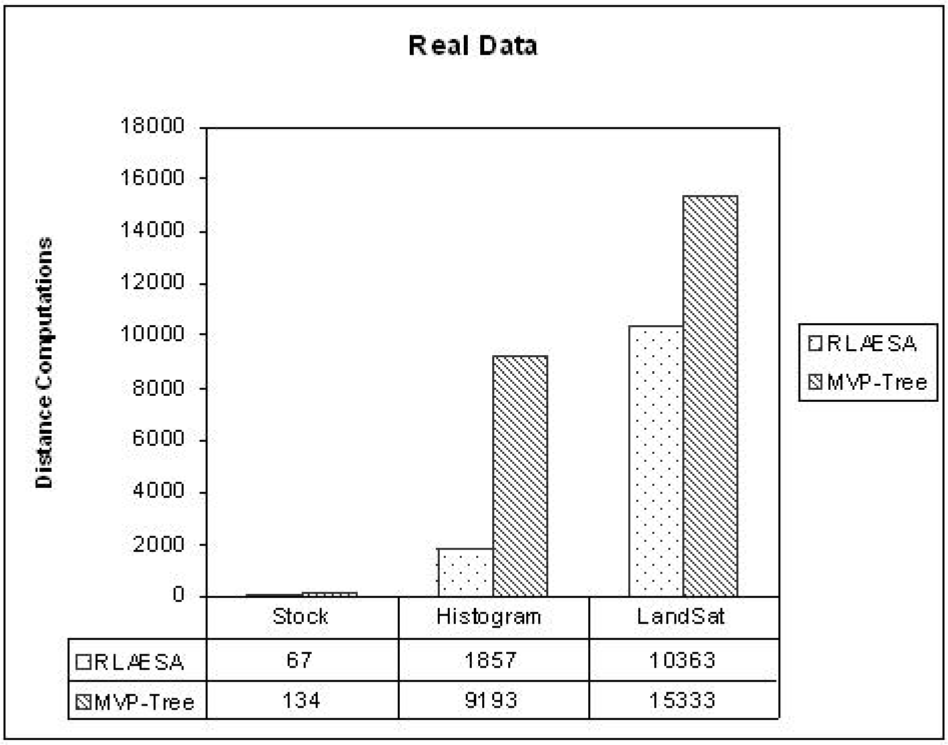
Figure 4.5: Comparison of grouping methods.

respectively. Since $g = n/k$, the number of the distance computations for method 1 will be $O(n^2/k + kn)$. Thus method 1 is not suitable for large data sets.

4.3.5 Experimental Results

We first consider the three grouping methods in RLAESA. Artificial data with uniform distribution is constructed in a 10-dimensional hypercube with length 1 on each side. Each test is repeated five times. Figure 4.5 shows that method 1 has the best query performance, and the performances of methods 2 and 3 are almost equivalent. Thus grouping method 1 has best query performance, but as mentioned previously, its construction cost is quadratic in the data size. This leads to a trade off between query performance and construction cost. For large data sets, methods 2 and 3 have a better balance between query performance and construction cost. In the rest of this section, we therefore choose to use grouping method 2 only.

MVP-tree (MVPT) also uses all-at-once pivot selection and takes linear space, and it is a leading algorithm in similarity search. Chavez [8] reports that MVP-tree outper-



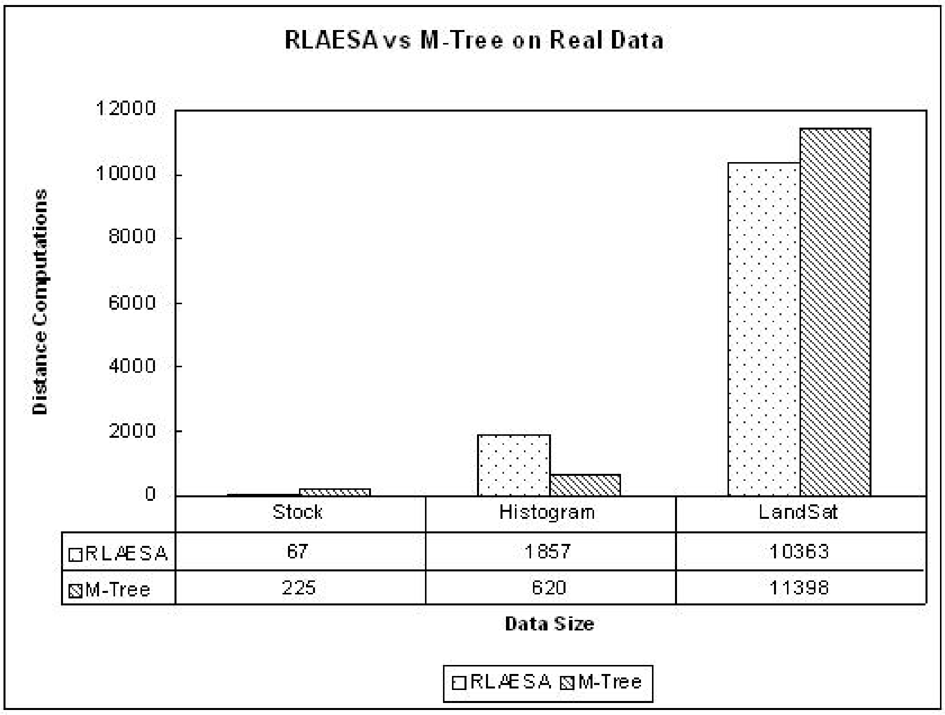
	Stock	Histogram	LandSat
□RLAESA	67	1857	10363
▨MVP-Tree	134	9193	15333

Figure 4.6: Performance on real data.

forms VP-tree (VPT) and GH-tree (GHT). Here we compare RLAESA with MVP-tree.

The three real data sets, stock, histogram, and land satellite, used in Section 4.2.2 is used for the final comparison. However, only some dimensions in the data sets are used to avoid excessive computation time. The first 30 dimensions of the stock data set are used, while the first 15 dimensions of Histogram and Land Satellite data sets are used. All experiments are repeated 5 times to get the average number of distance computations. Figure 4.6 shows that on all three data sets RLAESA requires significantly fewer distance computations than MVP-tree does.

As a matter of interest, we also compare RLAESA with M-tree [9], an alternative divide-and-conquer algorithm, on these three real data sets as above. Note that taking only the first 15 dimensions of the histogram and landsat makes them less clustered than before. Histogram data set is more clustered than landsat. Figure 4.7 shows that RLAESA works better on the scattered stock data set and first 15 dimensions of landsat



	Stock	Histogram	LandSat
□RLAESA	67	1857	10363
▨M-Tree	225	620	11398

Figure 4.7: Comparison of RLAESA with M-tree

data set, but worse on the clustered histogram data set as compared to M-tree.

The experimental results with RLAESA not only show it significantly outperforms MVP-tree, the leading algorithm using all-at-once pivot selection; but also supports our previous conclusion that all-at-once is preferrable for scattered data sets, while divide-and-conquer is better for clustered data sets.

4.4 NGHT

A variant of GHT (GH-tree) with improved pivot sharing is presented in this subsection. The relationship between distance computations and pivot sharing is analyzed via experiments. In the remainder of this thesis, we use NGH-tree (New gh-tree) or NGHT as the name for this index structure.

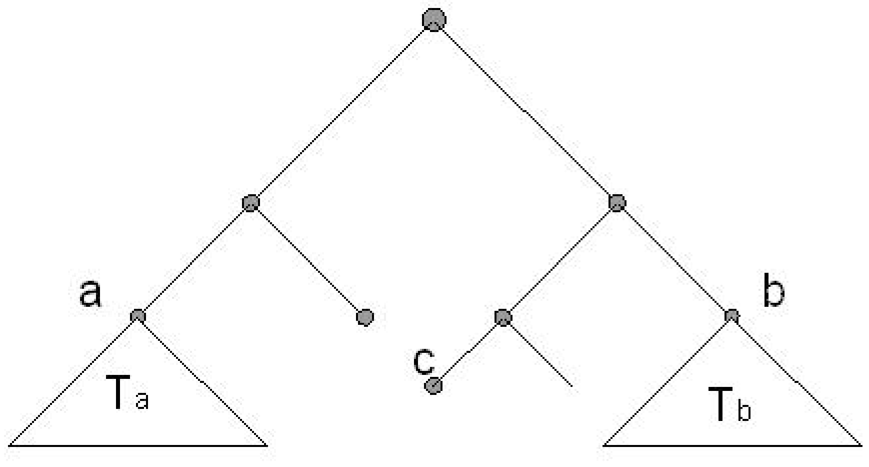


Figure 4.8: Sharing depth of NGHT

4.4.1 Construction Process for NGHT

The first step for NGHT is to construct a GHT, as described in Section 2.1.2.3 and in more detail elsewhere [22].

With a suitable integer for *sharing depth*, we adjust the pivot sharing level as follows: Consider two sibling nodes a and b at the same level inside GHT. If a and b have a common ancestor within depth sh, the distances from each object in the subtree T_a to the pivot of b are pre-computed and the pair of the maximum and the minimum distances is stored as the range of a with respect to the pivot of b. Similarly, the range of b with respect to the pivot of a is obtained and stored. Note that a and b can refer to the same node. A graphical explanation is provided in Figure 4.8. Let sh be 2. In this case, the pivot in node a will be used/shared by the objects in subtree T_b; similar for node b and subtree T_a. However, the pivot in node c will not be used/shared by all objects in *either* subtree T_a or T_b.

4.4.2 Search on NGH-tree

Search on NGHT is similar to search on GHT but more conditions are applied to tighten the lower-bound distances. Recall that the lower-bound distance on GHT is $|\frac{d(p_a,q)-d(p_b,q)}{2}|$ where p_a and p_b are the pivots to construct the hyperplane. Beside this, two more formulas are used to tighten the lower-bound distance.

- $d(q, p_a) - max(a)$ where p_a and $\max(a)$ are the pivot and the maximum distance for node a, respectively. Moreover, $d(q, p_a) > \max(a)$ must hold for this formula to be used.

- $\min(a) - d(q, p_a)$ where $p_a a$ and $\min(a)$ are the pivot and the minimum distance for node a, respectively. Moreover, $\min(a) > d(q, p_a)$ must hold for this formula to be used.

From all of the above three formulas, the one with the largest value is used as a new lower-bound distance for node a. Then this new lower-bound distance is applied in the same way as the lower-bound distance in GHT to prune away the subtrees which cannot contain the result object(s).

In short, search in NGHT requires fewer distance computations than that in GHT because more information is stored in order to provide tighter lower-bound distances during query time.

4.4.3 Analysis

Now we analyze the NGH-tree with respect to the space and construction cost.

4.4.3.1 Space Requirement

NGHT consists of a GHT structure and the additional range information. Each node in NGHT has at most 2^{sh-1} ranges, where sh is the sharing depth. Thus the range information in NGHT takes $O(2^{sh-1})$ times storage space as that in GHT. GHT has

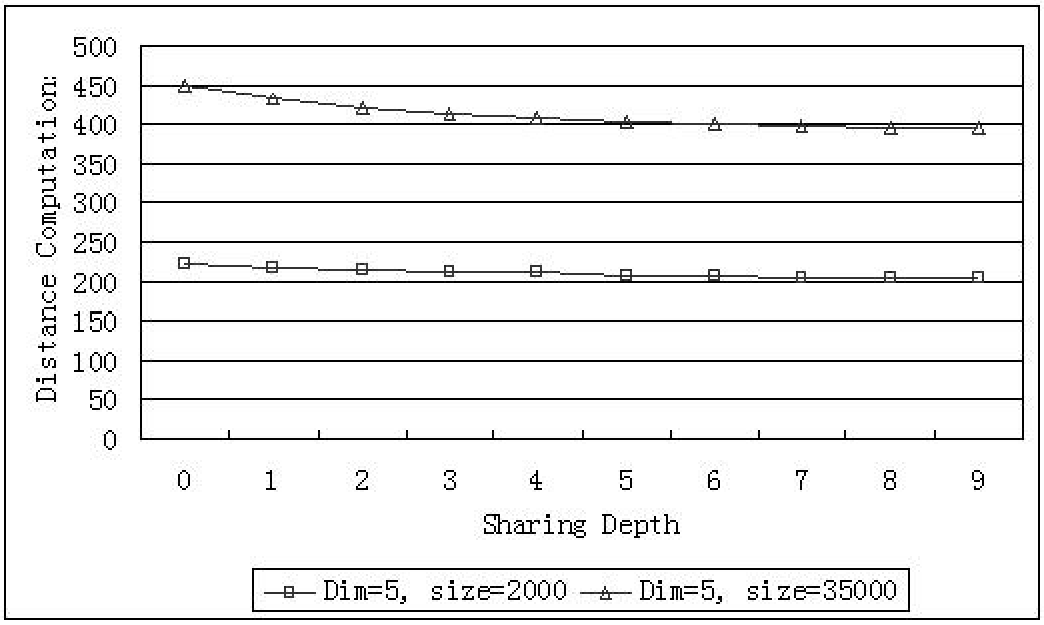


Figure 4.9: The relationship between sharing depth and distance computations for uniformly distributed data

linear storage space requirement in total. Thus, NGHT requires $O(n \cdot 2^{sh-1})$ storage space.

4.4.3.2 Construction Cost

In GHT the distances from the objects inside a node to the two pivots of the same node are pre-computed, while in NGHT the distances from the objects inside a node to the pivots of at most 2^{sh-1} nodes are pre-computed. The construction cost for GHT is $O(n \log n)$ distance computations. Thus, the construction cost for NGHT is $O(2^{sh-1} n \log n)$ distance computaions.

4.4.4 Experimenetal Results

Uniformly distributed data is used in the following experiments. In the first experiment, 2000 objects are distributed in a 15-dimensional hypercube with unit lengths. The results are shown in Figure 4.9. It can be seen that increasing the sharing depth reduces the number of distance computations at query time. Since the sharing depth

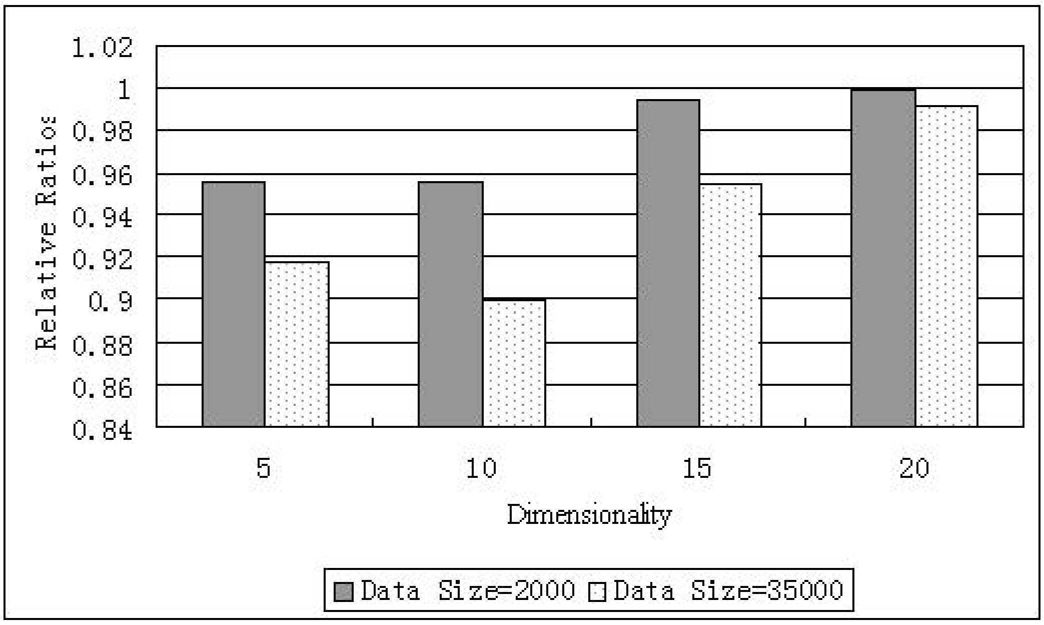


Figure 4.10: Relative Ratio of the number of distance computations for NGHT to GHT under different dimensionalities and different data sizes

defines the amount of information stored, the distance computations are saved at the expense of the cost of storage space. For different data sets, we are more interested on the percentage of distance computations being reduced than the absolute value. The number of distance computations with sharing depth of 3 is divided by that with sharing depth of 0 to obtain the *relative ratio* shown in Figure 4.10. It can be seen from the figure that the percentage of distance computations reduces for larger data sets. This is because each node in NGH-tree has a higher percentage of being pruned than that in GHT. If a node is pruned, the whole subtree rooted at it can be ignored for further consideration. For a while, the percentage of distance computation reduction grows as the data size increases. Moreover, it can be seen that the relative ratio is at a valley when the number of dimensions is 10 for a fixed data size. More investigation is necessary in the future to find out the reason.

NGHT demonstrates that distance computations in uniformly distributed data sets can be reduced by increasing pivot sharing, and this can be applied to a general data structure with minor modification only. Moreover, it is shown by experiment that its application is more effective for large data sets.

4.5 Conclusions

In this section, pivot sharing and pivot localization are defined and their relationship is evaluated through probabilistic analysis and experiments. We conclude that pivot sharing is preferrable for scattered data while pivot localization is better for clustered data if the storage space is fixed. Two data indexing structures, RLAESA and NGHT, are proposed to provide test data. RLAESA is a new data structure with linear storage requirement and it outperforms MVPT, the leading algorithm with all-at-once pivot selection. NGHT demonstrates a general framework which can be used to improve search efficiency by increasing the pivot sharing level.

Chapter 5

Reducing I/O

Most current similarity search algorithms focus on reducing the number of distance computations while paying little attention to I/O cost. We have followed this convention in the previous sections. However, as the data size grows, the percentage of information that can be loaded into main memory will be low. Depending on the data sizes and the applications, I/O cost might play an important role in query cost. In this section we try to reduce the I/O cost while keeping the number of distance computations low.

M-tree is the only indexing structure designed specifically for secondary memory [8]. The tree nodes in M-tree are stored in a single disk page. The insertion of new data object in M-tree is similar to that in B-tree. M-tree is balanced so that the I/O cost is low. It is not hard to imagine that other tree structures can be turned into this B-tree style [8]. For instance, the subtrees of a tree structure can be "compressed" into a disk page so as to take advantage of each disk read. However, it is hard to apply this style to non-tree structures such as AESA and LAESA.

In this chapter, a new indexing structure is proposed to alleviate the I/O problem for LAESA-like structures via a B-tree approach. We call this new structure SLAESA (sequential LAESA).

5.1 SLAESA

As in LAESA, k pivots $p_1, p_2, ..., p_k$ are chosen. For each i, compute the distances from p_i to each data object and store the objects into a double-linked B+-tree T_i. There are k double-linked B+-trees in total with the objects sorted by distances in the leaves. The construction requires $O(kn)$ distance computations and the storage cost is also $O(kn)$ where n is the data size as before.

By the triangle inequality, we have $d(q, o) > |d(q, p) - d(p, o)|$. Given a range query with query q and range r, each data object o within distance r to q satisfies $d(p_i, o) - r \leq d(q, o) \leq d(p_i, o) + r$ for each i. Thus all other objects violating the above conditions can be eliminated from further consideration. Let the data set be U, the object set satisfying $d(p_i, o) - r \leq d(q, o) \leq d(p_{iS}, o) + r$ be A_i and the object set violating $d(p_i, o) - r \leq d(q, o) \leq d(p_i, o) + r$ be B_i. We have $A_i = U - B_i$. We wish to find $\cap_{1 \leq i \leq k} A_i$ which can be obtained by starting with A_1 and iteratively intersecting with A_i or subtracting B_i, whichever is smaller. Since the distances inside the B+-trees are sorted, this operation takes $O(A_i)$ or $O(B_i)$ disk reads. Thus it takes $O(A_1 + \sum_{2 \leq i \leq k} \min A_i, B_i)$ disk reads on k B+-trees overall.

Nearest neighbour search in SLAESA is slightly different from range search due to the fact that the range distance is not available. Nearest neighbour search can be envisaged as gradually increasing A_i page by page. The details of the algorithm are as follows.

1. Compute all of the distances from the k pivots to the query.

2. Construct a priority queue which is minimum-oriented.

3. Construct an integer array V with n elements.

4. Store the k pivots into the priority queue using their distances as keys.

5. For each B+-tree T_i, read in the corresponding leaf node v_j where the pivot p_i is located. For each pair of T_i and v_j, insert pairs L_i, j and R_i, j into the priority

queue with lower-bound distances of the leftmost object and rightmost object of v_j, respectively, as the keys. For each object o in the node v_j, increment the value of $V[o]$ by 1.

   (a) If $V[o]$ equals k, insert o back into the priority queue with its lower-bound distance $|d(p_i, o) - d(p_i, q)|$.

6. Repeatedly retrieve the element with the smallest distance from the priority queue.

   (a) If the element is an object with its real distance computed, return it as the nearest neighbour.

   (b) If the element is an object with lower-bound distance, compute its distance to the query and insert it back to the priority queue with its real distance.

   (c) If the element is a pair of L_i, j, read in the leaf node v_{j-1} in tree T_i. Insert the pair $L_i, j-1$ back into the priority queue with the lower-bound distances of the leftmost object of v_j as the key. For each object o in the node, increment the value of $V[o]$ by 1.

      i. If $V[o]$ equals k, insert o back into the priority queue with its lower-bound distance $|d(p_i, o) - d(p_i, q)|$.

   (d) Otherwise, the element is a pair of R_i, j, read in the leaf node v_{j+1} in tree T_i. Insert the pair $R_i, j+1$ back into the priority queue with the lower-bound distances of the rightmost object of v_j as the key. For each object o in the node v_j, increment the value of $V[o]$ by 1.

      i. If $V[o]$ equals k, insert o back into the priority queue with its lower-bound distance $|d(p_i, o) - d(p_i, q)|$.

Now let's compare the search efficiency of SLAESA with LAESA. SLAESA has the same distance computations as LAESA since the pruning conditions using the triangle inequality are exact the same. The major difference between them are the I/O access cost. Let's assume the nearest neighbour searches with the same query object are run on both SLAESA and LAESA. For a specific object o, we assume that g out of k pivots are such that when they are used to obtain lower-bound distances, they lead to the elimination of o. The order of pivots being used in LAESA can be treated as random. By probability, it can be known that the average number of pivots is $\frac{k}{g}$ so that one of the g pivots is chosen and leads to the elimination of this object o. For SLAESA, the order of pivots being used is such that the pivot providing the smaller lower-bound distance will be used first. Thus, $k - g + 1$ are needed before the lower-bound distance leading to the elimination of o can be obatined. SLAESA needs to access almost g times more pivot-object distances than LAESA. However, in practice k for LAESA will seldom exceed 200 [16] because of the storage space restriction and g is some number smaller than k. Moreover, SLAESA uses sequential read while LAESA requires random access. One disk page in modern computer system can easily store more than 1000 pivot-object distances. Thus, SLAESA can easily outperform LAESA by 5 times in terms of I/O access.

At least two variants of SLAESA are available for different applications. The first variant follows. Note that because the distances are sorted in SLAESA, a single distance can be used for several close objects if precise lower-bound distances are not required. By doing so, fewer disk I/O operations are required. Of course, the number of distance computations will increase slightly since the lower-bound distances are no longer as tight as before. It is a tradeoff between CPU cost and disk I/O cost. The second variant tries to improve the order of pivots being used. Choose arbitrarily h out of k pivots and form h B+-trees as before. Form a distance matrix as in LAESA with the remaining $k - h$ pivots. By using this distance matrix, we start the nearest neighbour search as in LAESA. However, each time when a distance from an object to the query is computed, we know for sure that any object with lower-bound distances bigger than the above distance is not in the result and can be eliminated. By sequentially scanning the h B+-trees, we can mark some objects as being eliminated. If the same object is retrieved from the priority queue, it can be discarded. Note that in such a way, the tightest lower-bound distances are obtained in the early stage of the search. The implementation and testing of hese two variants are left for future work.

5.2 Experimental Results

Since the number of distance computations is no longer the only factor of search efficiency, we use the query time to evaluate the search efficiency. The experiments are performed on a PC with a 388 Hz CPU and a 256MB physical memory, running on a Redhat Linux 8 OS. The data set contains 100,000 artifical objects, distributed uniformly in a 10-dimensional hypercube. The numbers of pivots, k, in both LAESA and SLAESA are 100. Both indices take 80MB. Tests are repeated for 5 times on both SLAESA and LAESA. The average query time for SLAESA is 138 seconds while LAESA takes 147 seconds in average. Due to the long construction time, this is the only experimental results we have at this moment. More experiments are needed in further work.

5.3 Conclusions

For large data sets, it is necessary to consider not only the cost of distance computations but also that of I/O accesses. Experiments demonstrates that SLAESA outperforms LAESA in some large data sets. Further experimentation on different data sets and other variants of SLAESA are needed.

Chapter 6

Conclusions and Future Work

In this thesis, we analyzed pivot selection probabilistically. We proved that whether a pivot should be chosen from far away or within the cluster depends on the critical radius, r_c and obtained the value of r_c. Moreover we defined the concepts of pivot sharing and pivot localization and analyzed them and their relationship. We conclude that pivot sharing level should be increased for scattered data while pivot localization level should be increased for clustered data. This means that the algorithms using all-at-once pivot selection are preferrable for scattered data while those using divide-and-conquer pivot selectlion are better for clustered data.

Two new algorithms, RLAESA and NGH-tree, were proposed. RLAESA is an algorithm using all-at-once selection, and it outperforms MVP-tree, the fastest algorithm using all-at-once slelection and linear storage space. By studying the NGH-tree, we showed a general framework which can increase the pivot sharing levels for any algorithm using divide-and-conquer selection without decreasing the pivot localization level, so that the search efficiency can be improved. The experiments with RLAESA and NGH-tree not only show their performance but also support, with some other experiments, the above conclusion that pivot sharing level should be increased for scattered data while pivot localization level should be increased for clustered data.

Furthemore, we addressed disk I/O, which has received too little attention in most algorithms, and presented the SLAESA to reduce disk I/O cost.

A next step is to design an algorithm which is suitable for both scattered and clustered data. One approach is to build RLAESA on top of another algorithm using divide-and-conquer selection, since the data is usually scattered within clusters. It is also interesting to see how to combine these algorithms with the techniques of reducing disk I/O.

Bibliography

[1] R. Baeza-Yates, W. Cunto, U. Manber and S. Wu. Proximity matching using fixed-queries trees. Proceedings of the Fifth Combinatorial Pattern Matching (CPM'94), Lecture Notes in Computer Science, 807:198-212.

[2] R. Baeza-Yates and B. Ribeiro-Neto. Modern Information Retrieval Addison-Wesley, Reading, Mass, 1999.

[3] C. Bohm, S. Berchtold and D.A. Keim. Searching in high-dimensional spaces — Index structures for improving the performance of multimedia database, ACM Comput. Surv., 33(3):322-373, 2001.

[4] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. Proceedings of the 1997 ACM SIGMOD international conference on Management of data, 357-368, 1997.

[5] S. Brin. Near Neighbour Search in Large Metric Spaces. In Proceedings of the 21st Conference on Very Large Databases (VLDB'95), 574-584, 1995.

[6] W.A. Burkhard and R.M. Keller. Some Approaches to Best-Match File Searching. Commun. ACM, 16(4):230-236, 1973.

[7] E. Ch\'{a}vez and J. Marroquin. Proximity queries in metric spaces. Proceedings of 4th South American Workshop on String Processing (WSP'97), 21-36, 1997.

[8] E. Ch\'{a}vez, G. Navarro, R. Baeza-Yates and J. Marroqu\'{i}n. Searching in Metric Spaces. ACM Comput. Surv, 33:273-321, 2001.

[9] P. Ciaccia, M. Patella and P. Zezula. M-Tree: An efficient access method for similarity search in metric spaces. In Proceedings of the 23rd Conference on Very large Database (VLDB'97), 426-435, 1997.

[10] P. Ciaccia and M. Patella. Pac nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. Proceedings of of 17th International Conference on Data Engineering, 244-255, 2000.

[11] K. Clarkson. Nearest neighbor queries in metric spaces. Discrete Comput. Geom., 22(1):63-93, 1999.

[12] A. Gionis, P. Indyk and R. Motwani. Similarity search in high dimensions via hashing. Proceedings of 25th International Conference on Very Large Data Bases, 518-529, 1999.

[13] G. Hjaltason and H. Samet. Ranking in spatial databases. Proceedings of Fourth International Symposium on Large Spatial Databases, 83-95, 1995.

[14] G. Hjaltason and H. Samet. Incremental Similarity Search in Multimedia Databases. Proceeding of 1998 ACM SIGMOD international conference on management of data, 237-248, 1998.

[15] A.E. Lawrence. The volume of an n-dimensional hypersphere http://staffi.lboro.ac.uk/~coael/hypersphere.pdf

[16] 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 Recogn. Lett., 15:9-17, 1994

[17] L. Mico, J. Oncina and R. Carrasco. A Fast Branch & Bound Nearest Neighbour Classifier in Metric Spaces. Patt. Recog. Lett., 17:731-739, 1996.

[18] G. Navarro. Searching in Metric Spaces by Spatial Approximation In Pro. 6th South American Symposium on String Processing and Information Retrieval (SPIRE'99), pages 141-148. IEEE CS Press, 1999.

[19] G. Navarro. Searching in Metric Spaces by Spatial Approximation. VLDB Journal, Vol. 11, Issue 1, 28-46.

[20] M. Shapiro. The Choice of Reference Points in Best-Match File Searching. Commun. ACM, 20(5):339-343, 1977.

[21] E. Tuncel, H. Ferhatosmanoglu and K. Rose. VQ-index: an index structure for similarity search in multimedia databases. Proceedings of the 10th ACM International Conference on Multimedia, 543-552, 2002.

[22] J. Uhlmann. Satisfying general proximity/similarity queries with metric trees. Inf. Proc. Lett., 40(4):175-179, 1991.

[23] J.K. Uhlmann. Metric trees. Applied Mathematics Letters , 4(5), 1991.

[24] E. Vidal. An algorithm for finding nearest neighbours in (approximately) constant average time. Pattern Recogn. Lett. 4(3):145-157, 1987.

[25] E. Vidal and M.J. Lloret. Fast speaker independent DTW recognition of isolated words using metric space search algorithm (AESA). Speech Communication 7:417-422, 1988.

[26] E. Vidal. New formulation and improvements of the Nearest-Neighbour Approximating and Eliminating Search Algorithm (AESA). Pattern Recognition Lett., 15:1-7, 1994.

[27] T.L. Wang and D. Shasha. Query processing for distance metrics. In Proceedings of the 16th International Conference on Very Large Databases (VLDB), 602-613, 1990.

[28] Y. Wen. Incremental Similarity Search with Distance Matrix Methods. University of Waterloo Course Project, 2003.

[29] P. Yianilos. Data Structures and Algorithm for Nearest Neighbor Search in General Metric Spaces. Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 1993.

[30] P. Yianilos. Excluded middle vantage point forests for nearest neighbor search. In DIMACS Implementation Challenge, ALENEX'99 (Baltimore, Md), 1999.

[31] Math World http://www.mathworld.com