Spring 2004

Efficient similarity search in high-dimensional data spaces

Yue Li
New Jersey Institute of Technology

ABSTRACT

EFFICIENT SIMILARITY SEARCH IN HIGH-DIMENSIONAL DATA SPACES

by
Yue Li

Similarity search in high-dimensional data spaces is a popular paradigm for many modern database applications, such as content based image retrieval, time series analysis in financial and marketing databases, and data mining. Objects are represented as high-dimensional points or vectors based on their important features. Object similarity is then measured by the distance between feature vectors and similarity search is implemented via range queries or k-Nearest Neighbor (k-NN) queries.

Implementing k-NN queries via a sequential scan of large tables of feature vectors is computationally expensive. Building multi-dimensional indexes on the feature vectors for k-NN search also tends to be unsatisfactory when the dimensionality is high. This is due to the poor index performance caused by the dimensionality curse.

Dimensionality reduction using the Singular Value Decomposition method is the approach adopted in this study to deal with high-dimensional data. Noting that for many real-world datasets, data distribution tends to be heterogeneous, dimensionality reduction on the entire dataset may cause a significant loss of information. More efficient representation is sought by clustering the data into homogeneous subsets of points, and applying dimensionality reduction to each cluster respectively, i.e., utilizing local rather than global dimensionality reduction.

The thesis deals with the improvement of the efficiency of query processing associated with local dimensionality reduction methods, such as the Clustering and Singular Value Decomposition (CSVD) and the Local Dimensionality Reduction (LDR) methods. Variations in the implementation of CSVD are considered and the two

methods are compared from the viewpoint of the compression ratio, CPU time, and retrieval efficiency.

An exact k-NN algorithm is presented for local dimensionality reduction methods by extending an existing multi-step k-NN search algorithm, which is designed for global dimensionality reduction. Experimental results show that the new method requires less CPU time than the approximate method proposed original for CSVD at a comparable level of accuracy.

Optimal subspace dimensionality reduction has the intent of minimizing total query cost. The problem is complicated in that each cluster can retain a different number of dimensions. A hybrid method is presented, combining the best features of the CSVD and LDR methods, to find optimal subspace dimensionalities for clusters generated by local dimensionality reduction methods. The experiments show that the proposed method works well for both real-world datasets and synthetic datasets.

EFFICIENT SIMILARITY SEARCH IN HIGH-DIMENSIONAL DATA SPACES

by
Yue Li

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

May 2004

APPROVAL PAGE

EFFICIENT SIMILARITY SEARCH IN HIGH-DIMENSIONAL DATA SPACE

Yue Li

Dr. Alexander Thomasian, Dissertation Advisor — Date
Professor of Computer Science, NJIT

Dr. Joseph Leung, Committee Member — Date
Distinguished Professor of Computer Science, NJIT

Dr. Wojciech Rytter, Committee Member — Date
Professor of Computer Science, NJIT

Dr. Vincent Oria, Committee Member — Date
Assistant Professor of Computer Science, NJIT

Dr. Jian Yang, Committee Member — Date
Assistant Professor of Industrial and Manufacturing Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Yue Li

Degree: Doctor of Philosophy

Date: May 2004

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2004
- Master of Science in Computer Science,
  Shandong University, Jinan, P.R. China, 1999
- Bachelor of Science in Computer Science,
  Shandong University, Jinan, P.R. China, 1993

Major: Computer Science

Presentations and Publications:

Y. Li, A. Thomasian, and L. Zhang, "Finding Optimal Subspace Dimensionality for k-NN Search in Clustered Datasets," *15th Int'l Conf. on Database and Expert Systems Applications (DEXA'04)* , submitted, 2004.

L. Zhang, A. Thomasian, and Y. Li, "A Persistent and Dynamic Ordered Partition Index for k Nearest Neighbor Search," *15th Int'l Conf. on Database and Expert Systems Applications (DEXA'04)* , submitted, 2004.

Y. Li, A. Thomasian, and L. Zhang, "An Exact k-NN Search Algorithm for CSVD," *IEEE Transactions on Knowledge and Data Engineering (TKDE) Journal*, in review, 2003.

To the memory of my mother.

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Alexander Thomasian, who not only served as my research advisor, providing valuable and countless resources, insight, and intuition, but also constantly gave me support, encouragement, and reassurance. Special thanks are given to Dr. Joseph Leung, Dr. Wojciech Rytter, Dr. Dr. Vincent Oria and Dr. Jian Yang for actively participating in my committee.

I would like to thank Dr. Byoung-Kee Yi and Dr. Vittorio Castelli for their help on my research. This work could not have been completed without their background help. Special thanks are given to Lijuan Zhang, who worked with me for two years and assisted me in implementing clustering and multi-dimensional indexing algorithms.

I would like to thank my other fellow PhD students in the Integrated Systems Laboratory – Chunqi Han, Chang Liu and Gang Fu – for their friendship, support, and assistance over years.

Finally, I would like to thank my husband, my father, my daughter, and my in-laws for their love and support.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Motivation

High-dimensional data has always been a challenge in many application areas, such as information retrieval, image processing, data mining, pattern recognition, and decision support. Contemporary Database Management Systems (DBMS) have become much more complicated than their predecessors and the term *database* does not only mean the traditional databases, such as relational or object-oriented databases, but also many other types of databases, such as

- **Multimedia databases:** Multimedia databases contain various types of data like images, audio and video clips. Multimedia databases are applied in a wide variety of fields [18] and among them, digital imagery plays a valuable role in numerous human activities. There are many applications dealing with photographic images, satellite images (or remotely sensed images [64, 53]), medical images (like 2-dimensional X-rays [62] and 3-dimensional MRI brain scans [6]), geologic images and biometric identification images (like finger printing [38]). In these applications, the goal is to find objects in the database that are similar to some target object. Therefore, each image is transformed into feature vectors by extracting features such as color, texture or shape with numeric values, and the "similarity" is actually determined by the *feature vectors* and distance measures between the vectors.

- **Time series databases:** Time series or time sequences data accounts for a major fraction of all financial, medical, marketing and scientific data and is usually used for analysis, data mining, and decision making. A time series is often called a *signal* [27]. Time series databases convert time series segments into

multi-dimensional points using some transformation such as Discrete Fourier Transform (DFT) [3] and Discrete Wavelet Transform [63]. Similarity search for time series, which is usually performed on transformed data, is very popular since people are interested in finding similar patterns in time series and the results of matching are often used for the further analysis of market trends.

- **DNA databases:** Genetic material (DNA) stores complete instructions for all the cellular functions of an organism. DNA is strings of a four-character alphabet, known as the nucleotide bases, represented by A, C, G, and T [78]. DNA databases contain a large collection of such long strings. A new string (e.g., an unknown disease) has to be matched against the old strings based on a certain distance function to find the best candidates.

The various types of databases all have feature vectors that can be represented as high-dimensional data points or vectors with numeric values, therefore they are referred to as high-dimensional datasets generally. Multi-dimensional databases require "Similarity Based" queries or Content Based Retrieval (CBR), rather than traditional queries which are based on keys. Searching for similar patterns in the above databases is essential because it helps in predictions, decision making, computer-aided medical diagnosis, hypothesis testing, and in data mining [27].

The similarity between two data objects is typically measured by the distance between two vectors and searching for objects thus becomes a search for points in the feature space. The choice of the distance metric is usually determined by the application. For different data and applications, the ways of mapping data objects to high-dimensional data points are different and so are the distance functions. Multimedia objects are represented by low-level features, such as spatial, shape, color histogram, and texture for images. Features are then transformed into high-dimensional points (vectors). For example, some applications focus on color features. In this case each object is represented by a 64-dimensional color histogram and the similarity

between two images is determined by the distance between the corresponding two color histograms. Besides color, some applications extract shape [39] or textures information [56], while others may require the mixture of color, texture, spatial and shape information.

Time series data are numerical in nature, but usually a signal is very long, e.g. the closing stock price for a whole year. The feature extraction methods for time series aim at approximating the original signal with a shorter one via some transformation. There are two types of matching: *whole match* and *sub-pattern matching*, where whole matching assumes that the data and query series have the same length and sub-pattern matching considers the more general case where the data and query series have different lengths.

The most commonly used distance metric is the Euclidean distance, which is also the default distance measure in this study. In Chapter 2, some of the other distance metrics are introduced.

There are several types of similarity queries, such as range queries, k-Nearest Neighbor (k-NN) queries, and spatial joins. The k-NN query is an important tool in CBR. It retrieves the k closest data objects to the query object. For example, in image databases, a typical query would be "*find 20 photographs most similar to a given photograph*", or "*locate 10 persons who have fingerprints most similar to that of the suspect*".

The traditional DBMS can hardly support such kind of queries because they do not have efficient access methods for multi-dimensional data. The B-tree [9] (or B+-tree) has been used as a classical indexing method for commercial databases, while it is a one-dimensional indexing method which is only suitable for traditional primary-key-search.

During the last twenty years, many multi-dimensional indexing structures have appeared, such as R-trees [36], k-d-trees [32], and grid files [59]. Multi-dimensional

indexes provide an efficient way to selectively access some data points in a large collection associatively. They work well in low-dimensional space. Unfortunately, as a result of the *dimensionality curse*, the efficiency of indexing structures degrades rapidly as the number of dimensions increases: almost all pages in an index have to be visited and the query processing is even slower than a sequential scan on the entire dataset [15]. In order to make existing multi-dimensional indexing methods suitable for high-dimensional data, many variants of the R-tree have been proposed, such as the R^+-tree [69], the R^*-tree [10], the SR-tree [42] and the X-tree [13]. The main idea is to improve the space utilization and minimize overlaps. Other methods improve the performance of multi-dimensional indexing methods by combining the advantages of two or more existing structures (like the hybrid tree [21]). These improvements could not solve the problem of the dimensionality curse when the dimensionality of a dataset is very high.

1.2 Challenges, Contributions and Outline

1.2.1 Dimensionality Reduction

A well-known technique to break the dimensionality curse is to reduce the dimensionality and then build a multi-dimensional index structure on the reduced dimensionality space. The challenge is to achieve index space compression with limited loss of information and in particular with little effect on information retrieval performance.

Dimensionality reduction methods are usually based on a linear or nonlinear transformation followed by retaining a subset of features which are supposed to be more *important*. Techniques based on linear transformations, such as the *Karhunen-Loeve Transform* (KLT), the *Singular Value Decomposition* (SVD) and the *Principal Component Analysis* (PCA) have been widely used for dimensionality reduction and data compression. The SVD is shown to be very effective in compressing large tables of numeric data. It relies on *global* information derived from all the vectors in a

dataset. Its applications are therefore more effective when the dataset consists of *homogeneously* distributed vectors. However, high-dimensional datasets in the real world are often not globally correlated. With *heterogeneously* distributed feature vectors, using SVD to perform dimensionality reduction on the entire dataset may cause a significant loss of information. In this case, data points are not globally correlated, but rather there exist subsets of data points which are locally-correlated. More efficient representation can be generated by dividing the dataset into clusters, reducing dimensionality individually or recursively. The two categories of dimensionality reduction techniques are classified as *global* methods and *local* methods.

Clustering and Singular Value Decomposition (CSVD) and *Local Dimensionality Reduction* (LDR) are two such local methods. CSVD clusters datasets using an off-the-shelf clustering method, rotates each cluster using SVD into an uncorrelated coordinating space, and then reduces the dimensionality. LDR identifies local correlated clusters with a special clustering technique and decide the subspace dimensionality for each cluster according to the correlations.

In this thesis, three methods for selecting dimensions to be retained for CSVD are presented and compared with each other, and then the best CSVD method is compared to LDR from the viewpoints of compression ratio, CPU cost and retrieval quality. Experiments are held on four datasets and the results show that CSVD outperforms LDR.

1.2.2 Exact k-NN Search

Methods for k-NN search can be divided into two categories: *exact* methods and *approximate* methods. Exact k-NN search returns the k closest points which are the same as the results obtained from linearly searching the original dataset, while approximate k-NN search returns approximate results and guarantees a certain accuracy (ratio of the number of relevant nearest neighbors to the total number of retrieved

points).

An algorithm to find the k-nearest neighbors has been proposed especially for CSVD. It is an approximate method since it violates the lower-bounding property [27]. Although the low-bounding property was initially stated for range queries, it also works for k-NN queries since a k-NN query can be transformed to a range query with an estimated search radius.

In this thesis, an exact k-NN algorithm is presented for local dimensionality reduction methods based on a multi-step k-NN search algorithm presented in [48]. Experiments with two datasets show that it costs less CPU time than the approximate algorithm at a comparable level of accuracy.

1.2.3 Optimal Subspace Dimensionality

Since dimensionality reduction results in distance information loss, the number of dimensions to be retained becomes a critical issue. Based on the lower-bounding property, the total cost of a similarity query should be the sum of index query cost and postprocessing cost, which is used to remove unqualified candidates. Reducing too few dimensions does not solve the problem of dimensionality curse, while reducing too many dimensions results in excessive distance information loss. There should be an optimum interval of dimensionality in which the performance is the best, i.e., the query cost is minimized. An optimal subspace dimensionality corresponding to the minimum query cost can be found for global dimensionality reduction through experiments or modeling. Local dimensionality reduction methods partition a dataset into multiple clusters, such that each of which has different subspace dimensionality. It is difficult to find the optimal subspace dimensionality for each cluster with respect to the total minimum query cost.

In this thesis a hybrid method is presented to discover the relationship among query cost, ratio of total information loss, and subspace dimensionality of each cluster

so that optimal subspace dimensionality can be determined. The experiments show that an optimal subspace dimensionality indeed exists and the proposed method works well for both real-world datasets and synthetic datasets.

1.2.4 Outline

The rest of the thesis is organized as follows. Chapter 2 provides a background on high-dimensional indexing techniques. Chapter 3 introduces the revised CSVD method and compares it to the LDR method [73]. Chapter 4 proposes an exact k-NN algorithm for local dimensionality reduction methods [51]. In Chapter 5, a hybrid method for identifying optimal subspace dimensionality with respect to minimum query cost is described in detail [52]. Finally, the conclusion is given in Chapter 6. In addition, some useful techniques and results are described in Appendixes, where a k-NN algorithm utilized in Chapter 5 is described in Appendix A, the CPU cost of two types of k-means algorithms is compared in Appendix B, and some background information about fractal dimensions as well as some experimental results related to a query cost model are given in Appendix C. Table 1.1 explains some of the symbols which are frequently used in this thesis.

Table 1.1 Symbol Table

Symbol	Definition
X	dataset or data matrix
Y	transformed or rotated data matrix
N	dimension, number of features or number of columns of X
M	number of points in the dataset or number of rows in X
x_i	the i-th point (vector) in X
$x_{i,j}$	the j-th dimension of i-th point (vector) in X
p	number of retained dimensions
Q	query object or point
$\vec{q}$	query vector
P	data object or point
$\vec{p}$	data vector
p_i	the value in i-th dimension of $\vec{p}$
H	number of clusters
C_h or X_h	cluster h
m_h	cardinality of C_h
n_h or $p^{(h)}$	subspace dimensionality of C_h
$\mu^{(h)}, R^{(h)}$	centroid and radius of cluster h
Λ	diagonal matrix of eigenvalues of a dataset
λ_i	the i-th eigenvalue of Λ
$\lambda_i^{(h)}$	eigenvalue associated with the i-dataset dimension of cluster h

CHAPTER 2

HIGH-DIMENSIONAL INDEXING

Over the last three decades, relational database management systems have been well developed and are now prevalent. Most of the contemporary databases are too large to fit in main memory and thus have to be stored on secondary storage – such as disks. Storing data on disks is important also because disk is non-volatile and provides highly reliable and durable storage. A major characteristic of the secondary storage is that it is organized into blocks (or *pages*). Accessing data from disk involves mechanical movement of the read/write heads of the disk, which is slow and expensive, therefore every disk access results in a whole block of data being brought into main memory. Thus, it makes a large performance difference if similar data are grouped into the same disk blocks.

Traditional access methods to handle query processing in databases are usually based on primary keys, which is one-dimensional, such as *hashing* or *B-trees*. However, this technique is not well suited to multimedia information retrieval, which is based on content similarity. Many multi-dimensional indexes have been proposed to support similarity search for multimedia and scientific databases [33]. When the dimensionality is very high, e.g., higher than 60 or even in hundreds, most of the multi-dimensional indexing methods have a poor performance because of the *dimensionality curse* [15, 76, 18]. One of the solutions to solve the problem is to do dimensionality reduction, then build an index on the dimensionality reduced data. Another solution is to map the high-dimensional points into one dimension by some special techniques and then build an index on the one-dimensional space. In the following sections, firstly some preliminary background information is given, then some important multi-dimensional techniques are described respectively.

2.1 Similarity Queries

Similarity queries can be classified as point queries, range queries, k-Nearest Neighbor (k-NN) queries, and spatial joins.

Point query: Find a data point in a dataset. Example: *given a query point, locate it in the database.*

Range query: Find all data points within a certain range (radius) ϵ to the query point. Example: *find all images showing a tumor of size less than 0.5cm.*

k-NN query: Among all data points, find the k points that are closest to the query point. Example: *find 20 images that are closest (or most similar) to the query image.*

Spatial join: Find all unique pairs of distinct data points, whose relative distance is less than a given radius ϵ. Example: *find all pairs of images that are closest (within distance 0.01) to each other.*

Range query and k-NN query have gained much more attentions than the others because, a point query can be described as a range query with $\epsilon = 0$, and spatial join is just like "all-pair nearest neighbor search".

The thesis concentrates on nearest neighbor queries for the reason that they play a central role in content-based retrieval from multimedia databases.

Based on the probability of the query distribution, there are two models for similarity queries [61]:

Random model, which assumes that the query points are uniformly distributed in the data space.

Biased model, which assumes that queries are more probable in high-density areas of address space, i.e., the queries and the data has the same distribution. This

is usually true in many applications, e.g., in a transportation application with a map of cities, one would expect few queries on deserts and bodies of water, and more queries on highly populated areas.

In this thesis, the sample queries are always extracted from the dataset, therefore they are *biased queries.*

It is easy to see that regardless of the type of query, the distance metric is very important. Different distance metrics have different functions to calculate distance between two points, and they make the meanings of "closest" or "range" quite different.

2.2 Distance Metrics

For a specific application based on a certain feature extraction method, the first important step is to provide a measure for the distance between two objects. In this thesis, $D(\vec{p}, \vec{q})$ is used to denote the distance of the two data points $\vec{p}$ and $\vec{q}$. There are many kinds of distance functions in the literature. For similarity search, $\mathcal{L}_P$ *norms* and *Quadratic distances* are two categories that are most useful.

$\mathcal{L}_P$-norms are a series of similarity measures defined as:

$$D_{\mathcal{L}_P}(\vec{p}, \vec{q}) = (\sum_{i=1}^{N} |p_i - q_i|^P)^{1/P}$$

When $P = 2$, it is the *Euclidean distance*, which is the most popular distance function. It is the default distance function ($D()$) in this thesis. $\mathcal{L}_1$ norm is also called *Manhattan distance* or *city block distance*, which is often used in GIS applications. In the extreme case, when $P = \infty$, the above function becomes *Maximum distance* and can be rewritten as:

$$D_{\mathcal{L}_\infty}(\vec{p}, \vec{q}) = \max_{i=1}^{N} |p_i - q_i|$$

The relationship of $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_\infty$ is shown as a 2-dimensional example in Figure 2.1 (a), where $D_{\mathcal{L}_1}(\vec{p},\vec{q}) = b + c$, $D(\vec{p},\vec{q}) = D_{\mathcal{L}_2}(\vec{p},\vec{q}) = \sqrt{b^2+c^2} = a$, and $D_{\mathcal{L}_\infty}(\vec{p},\vec{q}) = b$ (provided that $b > c$). Figure 2.1 (b) illustrates a range query with radius ϵ under the above three distance metrics in 2-dimensional space. Any point $\vec{p}$ that satisfies $D_{\mathcal{L}_1}(\vec{p},\vec{q}) \leq \epsilon$ lies in the diamond; any point $\vec{p}$ that satisfies $D_{\mathcal{L}_2}(\vec{p},\vec{q}) \leq \epsilon$ lies in the circle; and any point $\vec{p}$ that satisfies $D_{\mathcal{L}_\infty}(\vec{p},\vec{q}) \leq \epsilon$ lies in the square.

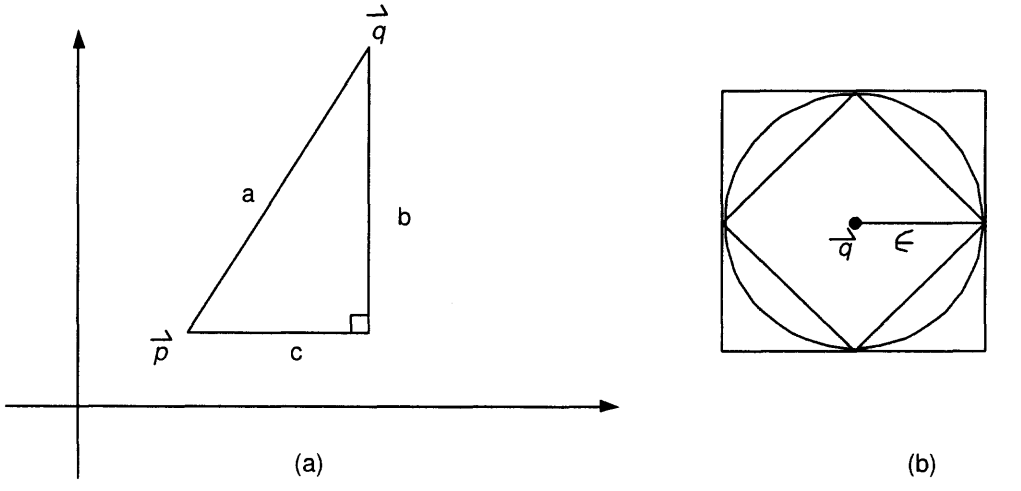


Figure 2.1 A 2-d example for (a) three types of distances (b) range query of radius ϵ under the three metrics.

Quadratic distances are weighted distance measures which are superior in CBR of multimedia objects, because they not only take account for the correspondence between each dimension as other distance metrics, but also make use of information across dimensions by capturing the correlation between dimensions [70, 81]. The quadratic distance between two feature vectors $\vec{p}$ and $\vec{q}$ is given by:

$$D_Q(\vec{p},\vec{q}) = (\vec{p}-\vec{q})^T A((\vec{p}-\vec{q})$$

where $A = [a_{i,j}]$ is an $N \times N$ matrix, and $[a_{i,j}]$ is the similarity coefficient between

dimension i and j. *Mahalanobis distance* is a special case of the quadratic distance metric in which the transform matrix is given by the covariance matrix obtained from a training set of feature vectors. The *normalized Mahalanobis* distance between vectors $\vec{p}$ and $\vec{q}$ is defined as [71]:

$$D_{Mahala}(\vec{p},\vec{q}) = \frac{1}{2}[N \ln 2\pi + \ln|C| + (\vec{p}-\vec{q})^T C^{-1}(\vec{p}-\vec{q})] \qquad (2.1)$$

where C is the covariance matrix and $|C|$ is the determinant of C. Actually it is a weighted Euclidean distance. It gives more weight to dimension with smaller variance and gives less weight to dimension with larger variance.

It depends on the application and feature extraction methods to select the distance metric to be used. For example, in time series analysis areas, Euclidean distance is often used. While in CBR, or image processing, Mahalanobis distance is widely used [71, 8, 40]. In Chapter 3, a high-dimensional clustering method – the *elliptical k-means* algorithm which utilizes Mahalanobis distance – is described.

2.3 Multi-dimensional Index Structures

Based on data types supported, multi-dimensional indexing methods can be classified into two broad categories: *Point Access Methods* (PAM) and *Spatial Access Methods* (SAM) [33]. PAM were primarily designed to perform spatial searches on point databases, which store only multidimensional points that do not have spatial extension. On the other hand SAM manage objects that, apart from their position in space, have spatial characteristics (shape). Such objects are lines, polygons, or higher-dimensional polyhedra. Since high-dimensional data are just points, and all SAMs can function as PAMs, it is not necessary to distinguish them in this study. Based on the partitioning of the data space, multi-dimensional index methods can be divided into *space-partitioning* methods, like grid files, k-d-trees [32] and quad-trees, which

divide the data space along pre-determined hyper-planes regardless of data clusters, and *data-partitioning* methods, like R-trees [36], X-tree [13], SR-tree [42], M-tree [24], and TV-tree [54], which partition the data space according to the data distribution.

Another classification of multi-dimensional access methods, which has gained more attention, is based on where the index resides: in main memory or on disk. Memory-resident multi-dimensional indexes (such as the k-d-tree) appeared a decade earlier than disk-resident methods (such as the R-tree). As the databases get larger and larger, disk-resident methods become more popular. In recent years, the sizes of main memories have increased rapidly and the prices have dropped rapidly, therefore large indexes can be held and queried in main memories easily. Consequently, memory resident indexes are popular again. For example, the ordered partition index [45] utilized in [19] is a memory resident index and it demonstrates a significant improvement over sequential scan.

The most popular indexing structures are discussed here and among them only k-d-trees are memory-resident methods and the others are disk-resident methods.

2.3.1 K-d-trees

The k-d-tree can be considered as an extension of the binary tree. It is an space-partitioning method which stores points in k-dimensional space. At each inner node, the k-d-tree divides the k-dimensional space in two parts by a (k-1)-dimensional hyperplane. The direction of the hyperplane alternates between the k possibilities from one tree level to the next. The coordinate axis can be selected using a round-robin criterion. Each splitting hyperplane contains at least one point, which is used as the hyperplanes representation in the tree. Points are stored at leaves. Searching and insertion of new nodes are straightforward. Deletion may cause re-organization of the tree under the deleted node, thus it is more complicated than insertion.

Since the k-d-trees are main memory data structures and they do not account

for paged secondary memory, they are not considered to be suitable for large spatial databases.

The k-d-B-tree [65] is a proposal to make the k-d-tree persistent. All nodes of the tree correspond to disk pages. A leaf node stores the data points that are located in the respective partition. Like the B-tree, the k-d-B-tree is perfectly balanced, however, it cannot ensure storage utilization. Figure 2.2 shows an example of k-d-B-tree in 2-dimensional space.

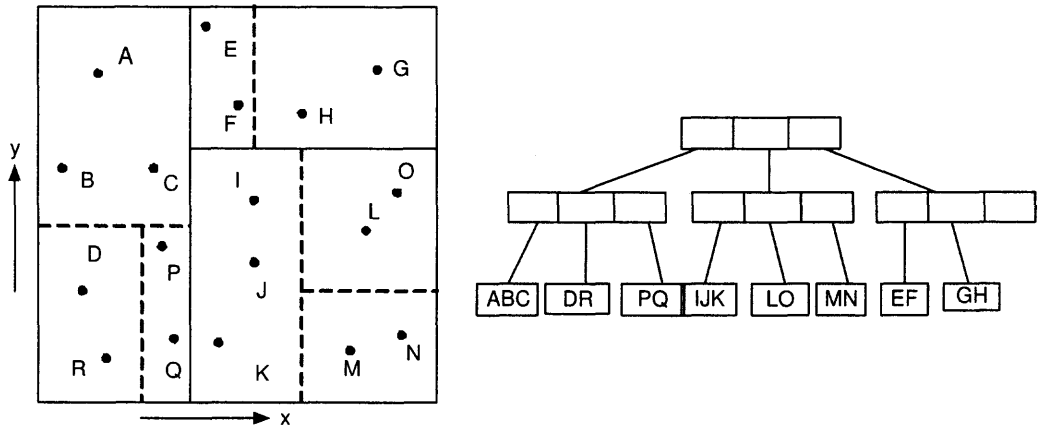


Figure 2.2 Example of a k-d-b-tree on 2-d space.

2.3.2 The R-tree and Its Variations

The R-tree is a hierarchical data structure with spatial extent. It is used to store not the original space objects, but rather their *Minimum Bounding Rectangles* (MBRs) which are defined as the minimum N-dimensional rectangle that contains the original N-dimensional object. The R-tree is balanced. Each non-leaf node contains entries with a form of *(ptr, Rect)*, where *ptr* is the address of a child node and *Rect* is the MBR of the child node. Leaf nodes contain entries of form *(obj_id, Rect)*, where *obj_id* points to a data object, and *Rect* is the MBR of that object. Figure 2.3 is an R-tree of height 2 in 2-dimensional space. The rectangles with dotted lines are MBRs. It is easy to see that this structure allows overlap among nodes.

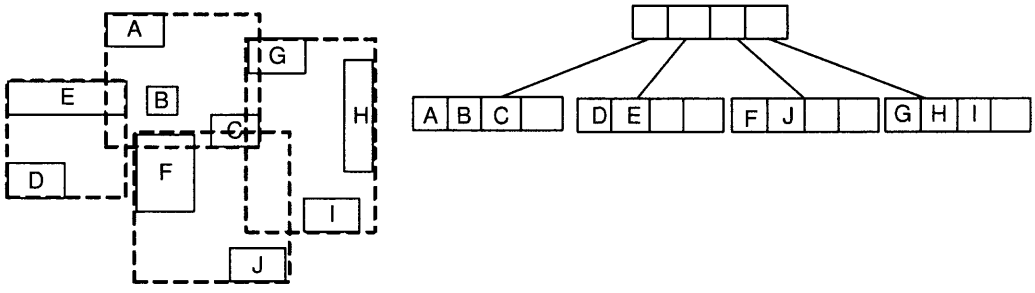


Figure 2.3 Example of a 2-d R-tree.

Searching in an R-tree is done from the root node in a *top-down* manner. All rectangles that intersect with the query object are visited. The R-tree does not guarantee that traversing one path of the tree is enough when searching for an object, since the MBRs may overlap one another. In the worst case, the search algorithm may have to visit all index pages for a query.

Insertion operation includes inserting the MBR of an object to the leaf node of the R-tree along with a reference to that object. If the MBR of the object intersects many entries of an intermediate node, the child whose MBR is enlarged least after the insertion will be selected. If the insertion causes the leaf page to overflow, the page splits in two. The split can be propagated to the ancestor nodes. If an insertion causes enlargement of the leaf page's MBR, it is adjusted properly and the change is propagated upwards.

Deletion in an R-tree requires an exact match query for the object. If the object is found in a leaf, it is deleted. The deletion may cause the leaf page to underflow. In this case the whole node is deleted, and all its entries are stored in a temporary buffer, and are reinserted in the tree.

There are many variations of R-tree, such as R^+-tree [69], which addresses the problem of minimizing overlap and R^*-tree [10], which introduces *deferred splitting* and *re-inserting* to improve the space utilization and minimize overlaps. Variations of R^*-tree include SS-tree [77] and SR-tree. The SS-tree partitions search space

into hyper-spheres rather than hyper-rectangles. The advantage of the SS-tree is that it requires much less storage compared to the R^*-tree because a hyper-sphere is determined by the center and radius, while a hyper-rectangle is determined by upper and lower bound of every dimension. However, the hyper-sphere occupies much larger volume than the hyper-rectangle with high-dimensional data and this reduces the search efficiency. The SR-tree utilizes both hyper-rectangles and hyper-spheres and uses the intersection of a bounding rectangle with a bounding sphere as the partitioning element. Compared to the R^*-tree and SS-tree, the SR-tree improves performance by saving more CPU time and disk accesses because it takes advantage of the good features of both methods above. Therefore in this study, the SR-tree is used for most cases when multi-dimensional indexing is involved. The X-tree is also an extension of R-tree, which introduces a more sophisticated split algorithm and *supernodes* in order to reduce the overlap. However, it is not easy to keep the tree balanced, also the large supernodes complicate the concurrency control mechanism.

The R-tree performs well for low-dimensional datasets since it is designed for spatial datasets with $2 \sim 3$ dimensions. Its variations improve performance by reducing overlap and improve storage utilization, but still can not make it efficient for high-dimensional datasets. In this study, high-dimensional datasets generally refer to datasets that have dimensionality more than 30. When the dimensionality becomes high, the performance of R-trees degrades because of the *dimensionality curse* and almost all data blocks need to be accessed. A number of recent results have shown the negative effects of increasing dimensionality on index structures [15, 76].

2.4 Dimensionality Curse

Dimensionality curse illustrates the problems caused by high-dimensionality. Human intuition based on experience with 3-dimensional world they live leads them to believe that numerous geometric properties hold in high-dimensional space, while in reality

they are not true in many cases.

Dimensionality curse affects multi-dimensional indexing and similarity search in two ways. One problem is related to *boundary effects*. For example, in 2 dimensional space, a circle with radius r is well approximated by the minimum bounding square (see Figure 2.4). The ratio of the areas of the circle to that of the square is $\frac{\pi r^2}{4r^2} = \pi/4 \approx 0.785$. However, in 100-dimensional space, the ratio of volume of the hyper-sphere to that of the minimum bounding hyper-cube becomes $\frac{\pi^{50}}{50! \cdot 2^{100}} \approx 1.87 \times 10^{-70}$. In this case the minimum bounding hyper-cube is an very poor approximation of the hyper-sphere since most of the volume of the hyper-cube is outside the hyper-sphere.

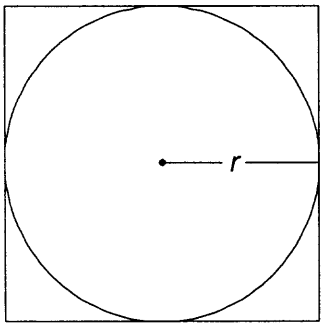


Figure 2.4 A circle and its minimum bounding square in 2-d space.

It is known that most of the multi-dimensional indexing structures partition data space into hyper-cubes or hyper-rectangles, therefore another example related to hyper-cubes is given here. Consider an 100-dimensional unit hyper-cube to be the search space and 100,000 points. Do a two-way-partition on each dimension and finally there are 2^{100} units. Even the data points are evenly distributed, most of the units (around 10^{25} units) are empty. It means for high dimensional data, the storage utilization is quite poor if the index is based on partitioning of the search space.

High dimensionality results in high overlap and low fan-out and therefore increase the number of page accesses for a query. When dimensionality is higher than a certain value, sequential scan can outperform a multi-dimensional index.

The other problem is intrinsic in the geometry of high-dimensional space. One

of the characteristic of high-dimensional space is that points randomly sampled from the same distribution appear uniformly from each other, and each point sees itself as an outlier [15, 1]. This makes the meaning of nearest neighbors questionable because, for each data point in a high-dimensional dataset, most of the other points have similar distances to it and the difference between the nearest neighbor and farthest neighbor is small. In Figure 2.5, the distances from each point to the centroid are recorded for all four datasets. It is easy to see that the data distribution of most of them follow normal distribution. The distances of most of the points to the centroid are similar to the average distance, e.g., for TXT55 (Figure 2.5 (b)), around 60% points are within a distance of 5 to the centroid while the average distance is 2.4. In this case, the difference between the 20th nearest neighbor and 40th nearest neighbor can be very small. In Figure Figure 2.5 (d), the nearest point from the centroid has distance 3.33 and the maximum distance is just 5.9 and there is almost no points within a range of 3 to the center. In this case, a range query becomes very sensitive to the choice of query radius: with a radius of 3, no result is returned; while with a radius of 4, 1/3 of points will be returned.

There are many techniques aimed at solving the problem of dimensionality curse. Dimensionality reduction is one of the most popular techniques, and also some new index structures or the revised version of the existing indexes are designed especially for high-dimensional applications. In addition, another category of indexing method called *Metric-Space Methods* or it Metric Based Indexing tries to index the distance between a data point and a query point rather than the data point itself.

2.5 Dimensionality Reduction

To reduce the dimensionality and then build a multi-dimensional index on dimensionality reduced data is a common technique to overcome the dimensionality curse. Usually for different applications, different dimensionality reduction techniques are

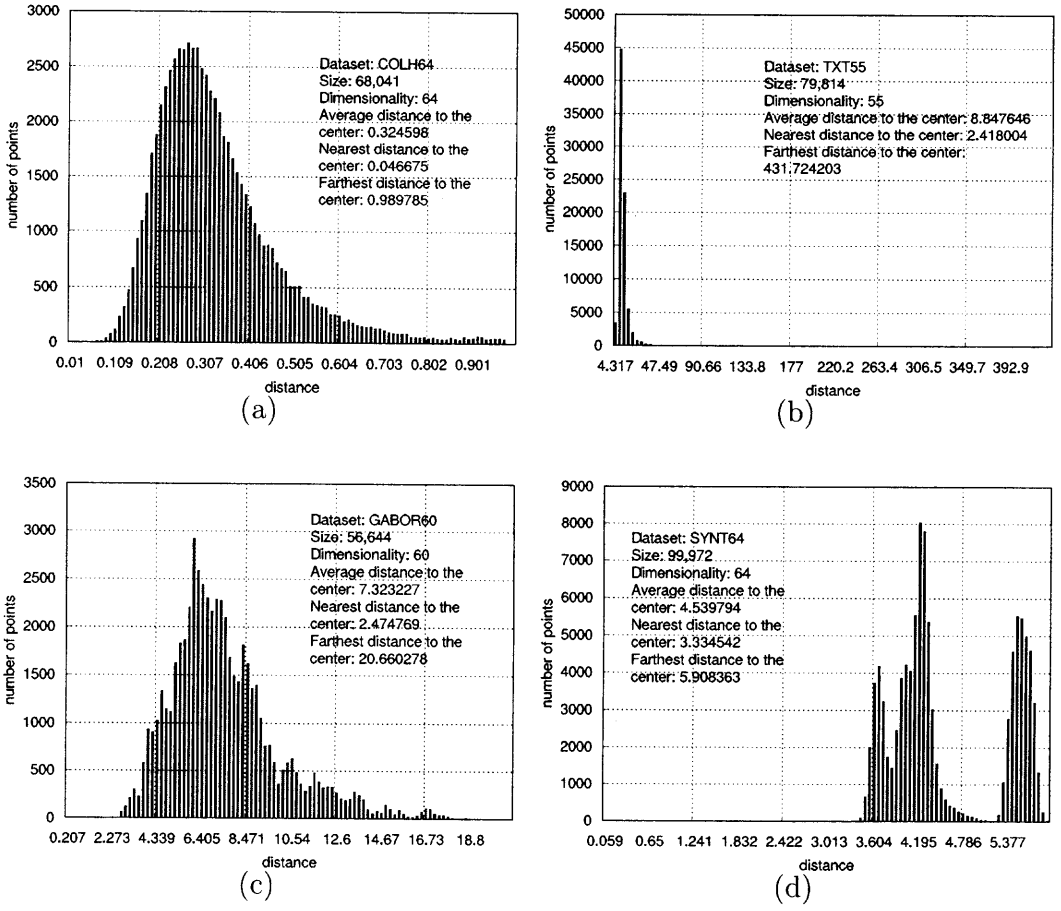


Figure 2.5 Distances between data points and centers of the datasets: (a) COLH64 (b) TXT55 (c) GABOR64 (d) SYHT64.

used. *Singular Value Decomposition* SVD or *Karhunen-Loeve Transform* (KLT), or *Principal Component Analysis* (PCA) [41]) is a most commonly used linear transformation method. SVD transforms a dataset by rotating and eliminating the correlations among dimensions. It can be used in many areas, especially image databases and it is "optimal" because it minimizes the *Normalized Mean Square Error* (NMSE) [27](see Section 3 for details). There are some other techniques which are used for time series databases, such as Discrete Fourier Transform (DFT), Discrete Cosine Transform (DCT) [34], Discrete Wavelet Transform (DWT) [63] and Segment Mean Method (SMM) [79]. DFT, DCT and DWT are also known as popular feature

extraction methods for image databases and signal processing [60].

The SVD, KLT and PCA reduce dimensions based on the global information of the whole dataset and they involve matrix calculation, therefore for applications which have large volume data and extremely high dimensionality, such as large time series databases, the DFT, DCT, DWT and SMM are applicable.

In order to be used for indexing, a dimensionality reduction technique must satisfy the *Lower Bounding Property* [27].

2.5.1 Lower Bounding Property

Dimensionality reduction results in distance information loss. There are two types of errors caused by dimensionality reduction and distance metric selection: *false dismissals* (or false negatives) and false alarms (or false positives). False dismissals refers to those qualifying objects that are not included in the set of retrieved objects, whereas false alarms refers to those non-qualifying objects that are included in the set of retrieved objects. For similarity queries on dimensionality reduced data space, in order to get correct answers, false dismissal should be avoided, otherwise it is not known for sure how many points in the response set are correct and that makes your answer meaningless. False alarms are acceptable since at least it guarantees all correct points are in the answer set, and furthermore, false alarms can be removed by post-processing as described in detail in Chapters 4 and 5.

Lemma 2.5.1 Lower Bounding Property (LBP) [27]: *To guarantee no false dismissals for searching in dimensionality reduced space (subspace), the distance between data points in subspace* $D^{(n)}()$ *must lower bound the distance between points in original space* $D^{(N)}()$.

If the distance function in subspace is just the Euclidean distance, the LBP holds because subspace dimensionality n is always less than or equal to the original dimensionality N. When a certain number of dimensions are removed, the difference

between $D^{(n)}()$ and $D^{(N)}()$ might be very large for some points. In order to have a better approximation of original distance, some revised subspace distance functions have been used like *approximate distance* in [19] and *NewImage()* in [22]. Only those that have the property of LBP can be used for exact k-NN search algorithms.

2.5.2 Normalized Mean Square Error

Before introducing any specific dimensionality reduction method, it is necessary to introduce a metric to measure the information loss due to dimensionality reduction. The Normalized Mean Squared Error (NMSE) is defined as the ratio of total distance information loss after dimensionality reduction to the total distance information before dimensionality reduction.

Given a matrix X (corresponding to dataset X) with size M and dimensionality N, if the transformed data matrix is X' in the original space, the definition of NMSE is as in Equation 2.2

$$NMSE = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}(x_{i,j} - x'_{i,j})^2}{\sum_{i=1}^{M}\sum_{j=1}^{N}(x_{i,j} - \mu_j)^2} \tag{2.2}$$

where μ_j is the mean of j-th column.

2.5.3 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a popular technique that has been used in numerous applications such as statistical analysis (as *Principal Component Analysis*), text retrieval (as *Latent Semantic Indexing*) and pattern recognition (as *Karhunen-Loeve transform*).

Let $X = [x_{i,j}]$ be an $M \times N$ matrix. Without loss of generality, it is assumed that the mean (average of each column) is zero. The SVD of X is the factorization

$$X = USV^T, \tag{2.3}$$

where U is an $M \times N$ column-orthonormal matrix, V is an $N \times N$ unitary matrix of *eigenvectors*, and S is a diagonal matrix containing the *singular values*: $s_1 \geq s_2 \geq ... \geq s_N$. When N' of the singular values are negligibly small, then the rank of X is $N - N'$.

Principal Component Analysis (PCA) is based on the decomposition of the *covariance matrix* C for X:

$$C = X^T X/M = V \Lambda V^T \tag{2.4}$$

V corresponds to the matrix of eigenvectors (as before) and Λ is a diagonal matrix which holds the *eigenvalues* of C. C is positive-semidefinite, hence its N eigenvectors are orthonormal and its eigenvalues are nonnegative. The trace (sum of eigenvalues) of C is invariant under rotation. Eigenvalues, similarly to the singular values, are in decreasing order. The singular values are related to the eigenvalues by: $\lambda_j = s_j^2/M$ or $s_j = \sqrt{M\lambda_j}$. The eigenvectors constitute the principal components of X, hence the transformation

$$Y = XV \tag{2.5}$$

yields uncorrelated features. Retaining the first p dimensions of Y minimizes the NMSE.

Since SVD is a linear transformation, it does not change the Euclidean distance between two arbitrary points. The distance information loss of transformed dataset is equal to that of the original dataset. Therefore after dimensionality reduction, the NMSE can also be defined as in Equation 2.6, where y_i is the i-th transformed data point and n is the number of dimensions retained, and zero-mean precondition still holds.

$$NMSE = \frac{\sum_{i=1}^{M} \sum_{j=n+1}^{N} y_{i,j}^2}{\sum_{i=1}^{M} \sum_{j=1}^{N} y_{i,j}^2} \tag{2.6}$$

There is a simpler way to calculate NMSE. According to Equation 2.7, the NMSE is equal to ratio of the sum of discarded eigenvalues to the sum of all eigenvalues, where λ_j is the j-th largest eigenvalue.

$$NMSE = \frac{\sum_{j=n+1}^{N} \lambda_j}{\sum_{j=1}^{N} \lambda_j} \tag{2.7}$$

Proof of Equation 2.7: Equation 2.8 can be obtained from Equation 2.5:

$$YV^T = X \tag{2.8}$$

Based on to Equation 2.4 and Equation 2.8,

$$C = X^T X/M = VY^T YV^T/M = V\Lambda V^T$$

By pre-multiplying and post-multiplying with V^T and V

$$Y^T Y = M\Lambda$$

Therefore Equation 2.7 holds.

It follows that the PCA and SVD achieve the same goal. The PCA requires MN^2 multiplications to compute C and obtaining eigenvalues is an O (N^3) computation, while SVD is $O(MN^2)$.

2.5.4 Discrete Fourier Transform (DFT)

Discrete Fourier Transform is one of the earliest techniques for reducing the dimensions of time series [3]. The basic idea is that any signal can be represented by the superposition of a finite number of sinusoidal waves, where each wave is represented by a single complex number known as a *Fourier coefficient.* A time series represented in this way is said to be in the frequency domain. There are many advantages to

representing a time series in the frequency domain, the most important of which is dimensionality reduction. A signal can be decomposed into sine waves that can be recombined into the original signal. However, the last few coefficients that contribute little to the reconstructed signal can be discarded without much loss of information thereby producing dimensionality reduction.

One of the useful properties of the DFT is that it preserves the energy (square of the length) of the signal, and the other property is that the DFT also preserves the Euclidean distance and therefore the lower bounding property is satisfied. These good properties make the DFT a popular method for indexing and dimensionality reduction.

2.5.5 Discrete Wavelet Transform (DWT)

Wavelets are mathematical functions that represent data in terms of the sum and difference of a prototype function, called the basis function. Several DWT methods have been proposed [50]. They are different from the DFT in several important respects. One important difference is that wavelets are localized in time, i.e., each wavelet coefficient of a signal contributes to the reconstruction of a small portion of the object. This is in contrast to the DFT where each Fourier coefficient contributes to the reconstruction of each and every data point of the time series. This property of the DWT is useful for multi-resolution analysis of the data. The first few coefficients contain an overall, coarse approximation of the data, while additional coefficients can be imagined as zooming-in to areas of high detail.

The simplest DWT to describe and code is the *Haar wavelets*. It gives the sum and the difference of the left and right part of a signal, then it focuses recursively on each of the halves, and computes the difference of their two sub-halves, until it reaches an interval with one only sample in it.

2.5.6 Segmentation Mean Method (SMM)

Segmentation mean method, which is also called Piecewise Aggregate Approximation [44], refers to a group of feature extraction methods that divide each time sequence into equal [79] or nonequal [20] sized segments and record the mean value of each segment as a feature. See Figure 2.6 for an example of equal sized SMM. They can be used for arbitrary $\mathcal{L}_p$ norms and it is very simple, therefore multiple similarity models can be supported. Also it guarantees "no false dismissal" since the distance in SMM space lower bounds the distance in the original space. However, this type of dimensionality reduction method is usually only used on time series databases because for time series "time" is the variable and the series reflects the change based on time, therefore it is still meaningful if the time interval becomes larger. For other datasets, like image databases, it may be meaningless to take the mean of shape information and color information.

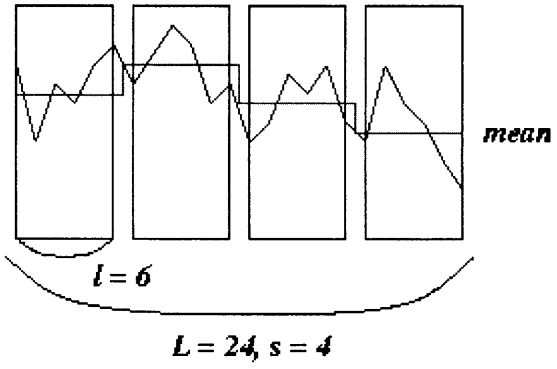


Figure 2.6 Segmentation mean method [79].

2.6 Metric Based Index Structures

Content-based retrieval of images, audio, video or other multimedia objects is usually based on the similarity between two objects. The basic idea is to extract features like shape, texture and color from multimedia objects and map them into high-

dimensional feature vectors. Then searching similar objects becomes searching feature vectors with smallest distances. The distance function can be very complicated for some application, while the most common one is the Euclidean distance ($\mathcal{L}_2$). The basic idea of metric based indexes is to take advantage of the properties of similarity to build a tree, which can be used to prune branches in processing the queries.

The M-tree is a height-balanced tree that uses both features and distance information for indexing [24]. It partitions multimedia objects on the basis of their relative distances and stores features of objects in leaf nodes, whereas it stores the so-called *routing nodes* as well as child node pointers in internal nodes. All objects in the subtree are within a certain distance (stored as an entry of the routing node) of the routing nodes. The routing nodes are mainly designed for pruning unnecessary traversal and checking. The insertion and splitting process is conceptually similar to the corresponding processes of the R-tree.

A more recently method called *iDistance* relies on partitioning the data and defining a reference point for each point [80]. Then each point is indexed using a single-dimensional value which is the distance to the reference point. iDistance is designed especially for k-NN search and its effectiveness depends on how the data are partitioned and how the reference points are selected.

Metric based indexing methods support fast similarity search and can solve some problems due to high-dimensionality, but they do not support range queries since data points are not indexed on individual attribute values.

2.7 Summary

In this chapter, some background information and techniques related to similarity search in high-dimensional space are given.

Generally, when dimensionality is not so high, multi-dimensional index structures can be used directly. If the dimensionality is very high, index structures might

be even less efficient than a sequential scan because of the dimensionality curse described in the chapter. Dimensionality reduction, followed by creating indexes on dimensionality-reduced datasets, can solve the problem. For datasets with *heterogeneously* distributed data, clustering should be performed before dimensionality reduction in order to minimize information loss. CSVD is selected as the basic method for dimensionality reduction in this study.

CHAPTER 3

PERFORMANCE OF CSVD

3.1 Introduction

Content Based Retrieval (CBR) or similarity search has been an area of intense interest in multimedia applications and it was given additionally impetus by the *QBIC (Query By Image Content)* project a decade ago [58]. Domain experts characterize objects by their features and define similarity measures, which can be incorporated into a *similarity index* [77]. CBR seeks to find objects in the database that are similar to some target object. Feature vectors and similarity measures for color, texture, and shape are discussed in Chapters 11, 12, 13 in [18], respectively. The objects are segments of an image for which feature vectors have been computed. Efficient k-NN search over feature vectors is concerned.

An object P is specified by its feature vector $\vec{p}$, which is a point or vector in N-dimensional space. N may be in the hundreds, while the number of objects M tends to be very large, e.g., in the millions. This information is summarized in a dataset X, which is in fact an $M \times N$ matrix.

The default distance metric used in this chapter is L_2, i.e., the *Euclidean distance*. For two points P and Q,

$$D(P,Q) = \| \vec{p} - \vec{q} \|_2 = \sqrt{\sum_{i=1}^{N} (p_i - q_i)^2}.$$

The processing of k-NN queries can be quite costly for large values of N and especially M, since it requires the scanning of the dataset to compute the distances of all points with respect to the query point Q. *Clustering* is one method to deal with this problem, see e.g., [30], since hopefully only the points in the appropriate cluster need to be considered in the processing of the k-NN query, although it is known that

clustering does not work as well as one would expect in high dimensional spaces.

Clustering can be attained indirectly by building a multi-dimensional index [33] to reduce the number of points to be checked. As reported in [77], as the dimensionality increases by a factor of 5-10, the performance of k-NN queries degrades by a factor of twenty for R-trees, as well as SS-trees described in [77].

Dimensionality reduction by rotating X to its principal components ($Y = XV$ as given in Chapter 2) and retaining p dimensions such that $p < N$ can be attained by applying a linear transformation commonly known as Singular Value Decomposition (SVD), or Karhunen-Loeve Transform (KLT) and performing Principal Component Analysis (PCA). PCA is the "best dimensionality reduction" method in that it minimizes the Normalized Mean Square Error (NMSE) (see Chapter 2 for definition) [27].

One of the earliest papers that utilize SVD in database area is [46]. It uses SVD to compress a large dataset into a format that supports ad hoc querying, provided that a small error can be tolerated when the data is uncompressed.

Ad hoc queries require access to data records, either individually or in the aggregate. Some of the typical queries are on specific cells of the data matrix like "*what was the amount of sales of ABC Inc. on December 2003*", others are aggregate queries on selected rows and columns like "*find the total sales of ABC Inc. for 2003*". To support such queries, one has to maintain "random access", i.e., fast reconstruction of any desired cell of the matrix. Thus the algorithm consists of 2-pass computation of SVD to get the eigenvectors (matrix V), eigenvalues (matrix Λ) (Equation 2.4), matrix U (Equation 2.3), and reconstruction matrix X', where $x'_{i,\,j}$ is the reconstructed value of any desired cell $x_{i,\,j}$ and

$$x'_{i,\,j} = \sum_{n=1}^{p} \lambda_n \cdot u_{i,\,n} \cdot v_{j,\,n} \qquad i = 1, ..., M;\ j = 1, ..., N \tag{3.1}$$

and p is the number of dimensions retained.

Some data points may be approximated poorly. For those cells for which SVD reconstruction shows the highest error, a set of triplets of the form (*row, column, delta*) is maintained, where *delta* is the difference between the actual value and reconstructed value of that cell. With this structure, one can adjust the query results to make them more accurate.

The intuition behind SVD can be illustrated in 2-dimensional space. Figure 3.1 gives an 2-d example to illustrate the rotation of axis that SVD implies. SVD transforms a sample of points to a best coordinate space along which the properties of the sample are most clearly exhibited. It can be seen that if only one dimension is retained, the "best" axis to project is x'.

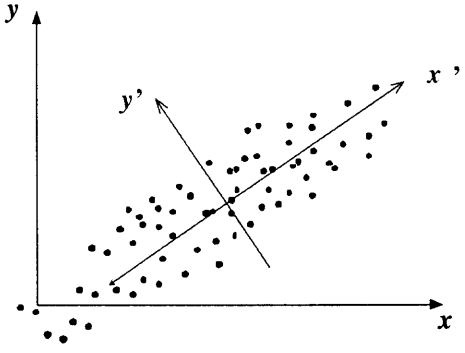


Figure 3.1 Example of SVD on a 2-d sample of points.

The SVD has been shown to be an efficient method for compressing large tables of numeric data. It relies on *global* information derived from the dataset, its application is therefore more effective when the dataset consists of *homogeneously distributed vectors*. In other words, SVD works well for a dataset whose distribution is well captured by the centroid and the covariance matrix [19]. For datasets with *heterogeneously* distributed vectors, however, performing dimensionality reduction using SVD on the entire dataset may cause a significant loss of information, as illustrated by Figure 3.2(a). In this case, data are not globally correlated, but rather subsets of data are locally-correlated.

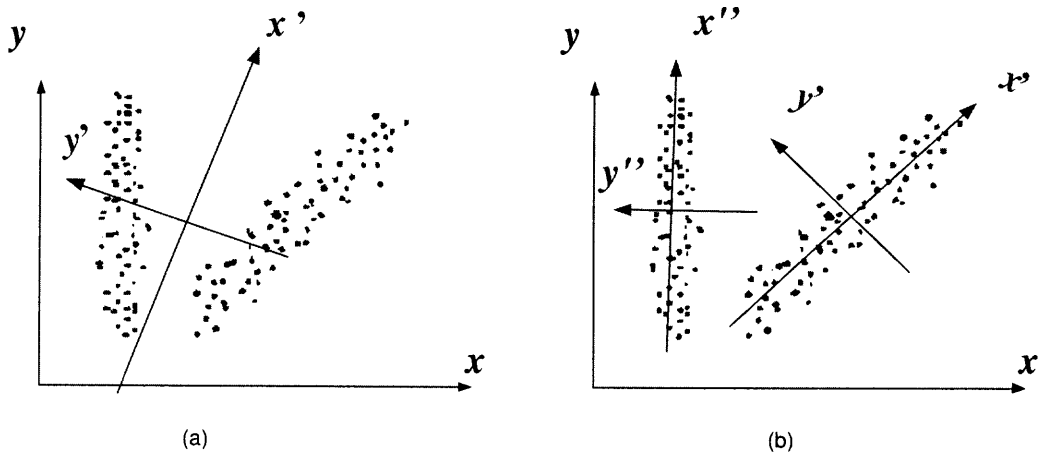


Figure 3.2 (a) SVD (b) CSVD on locally correlated data samples.

High-dimensional datasets in the real world are often not globally correlated and therefore a method that can capture the local correlations in a dataset is needed. The solution is to partition the large dataset into clusters, and do dimensionality reduction separately on each cluster (Figure 3.2(b)).

The *Clustered Singular Value Decomposition* (CSVD) method captures this local correlation by applying clustering first followed by SVD, although SVD can be optionally applied before clustering, which is an instance of *Recursive SVD - RCSVD* [72], [19]. This study is motivated by the Spire project at IBM Research, which applied CBR to texture feature vectors extracted from satellite images.

The *Local Dimensionality Reduction* (LDR) method [22] combines the dimensionality reduction step with clustering to seek clusters yielding a higher dimensionality reduction (or so called "SVD-friendly" clusters [72]). Another method called *Multi-level Mahalanobis-based Dimensionality Reduction* (MMDR) [40] tries to cluster a high dimensional dataset using the low-dimensional subspace based on *Mahalanobis* distance instead of Euclidean distance. The CSVD method is compared with LDR in [19], but additional results with new datasets are reported in this chapter.

Three variations for implementing CSVD are considered and compared against

each other, and also the SVD and LDR methods from the viewpoint of the compression ratio, CPU time, and retrieval efficiency. The comparisons are carried out using three real-world and one synthetic dataset [22]. Also an elliptical k-means algorithm based on Mahalanobis distance is implemented and compared with the regular k-means method from the viewpoint of compression ratio.

Two variations in merging query results obtained from different clusters are also evaluated, referred to as *on-the-fly* and *deferred* merging. In *on-the-fly* merging, the points retrieved from various clusters are merged based on their *approximate* distance, In *deferred merging*, each cluster yields k nearest neighbors, with *projected* or *approximate* distances, which can be merged using *original* distances.

The roadmap to this chapter is as follows. Section 3.2 provides a description on the related work, especially LDR method and MMDR method. Section 3.3 describes the steps required for implementing CSVD and Section 3.3.2 specifies the approximate algorithm for k-NN search. Section 3.4 summarizes experimental results for CSVD and LDR with and without indexing structures. Conclusions and areas for future research are given in Section 3.5.

3.2 Related Work

There are several methods in the literature that capitalize on the observation of doing dimensionality reduction based on *clustering*. Consequently, before those techniques are described, a brief introduction of clustering is needed.

3.2.1 Clustering

Clustering is the process of grouping a set of the objects into clusters where similar objects are aggregated into the same cluster and dissimilar objects into different clusters. Clustering is an unsupervised learning process, since it does not require any predefined training classes. As a branch of statistics, clustering plays an outstanding

role in information retrieval, machine learning and data mining applications such as scientific data exploration, text mining, spatial database applications, marketing, medical diagnostics, computational biology, and many other fields [57, 14].

Clustering algorithms can be classified into four categories: *Partitioning* methods (such as k-means [30] and EM algorithm [25]), *Hierarchical* methods (such as CURE [35] and BIRCH [82]), *Density-Based* methods (such as DBSCAN [26]) and *Grid-based* methods (such as STING [75]).

k-means is a commonly used partitioning clustering method. A regular k-means algorithm has the following steps:

1. Randomly select k points from the dataset to serve as centroids.
2. Assign each point to the closest centroid to form k clusters.
3. Recompute the centroid for each cluster.
4. Go back to Step 2 until there is no new assignment.

The k-means method is utilized for CSVD because it is the simplest clustering method that works well for CSVD. Since the quality of the clusters varies significantly from run to run, the clustering procedure is repeated for 10 times and the result yielding the smallest *sum of squares* (SSQ) distance (C_h denotes the set of points in the h^{th} cluster) is selected.

$$SSQ = \sum_{h=1}^{H} \sum_{i \in C_h} \sum_{j=1}^{N} (x_{i,j} - \mu_j^{(h)})^2. \tag{3.2}$$

The choice of clustering algorithm is often determined by applications. The regular k-means method tends to discover clusters with spherical shapes. In applications like image processing and pattern recognition, it is often desirable to find natural clusters. Data points that are locally correlated should be grouped into one cluster. Therefore another clustering method that discover elliptical shaped

clusters is also utilized, which is called *elliptical k-means* algorithm [71]. It is based on an adaptively changing *normalized Mahalanobis* distance metric as shown in Equation 2.1.

The outline of the algorithm is as follows [71]:

1. Utilize Euclidean-based k-means algorithm to obtain k clusters.
2. Compute the covariance matrix for each cluster.
3. Reassign points to the closest clusters based on the normalized *Mahalanobis* distances obtained with current k centroids and their covariance matrices.
4. If there is new assignment, recompute the k centroids and go back to step 3. Otherwise, go to step 5.
5. If the number of outer-loops (step 2 to 5) has not exceeded *MaxOutLoop*, go to step 2, otherwise return the current k clusters and their centroids.

The elliptical k-means algorithm differs from the regular k-means algorithm in that it iteratively refines and recovers the cluster shapes by recomputing the covariance matrix. Therefore, it ends up with the elliptical shaped clusters. However, there is a tradeoff for these advantage. The covariance matrix needs to be recalculated many times and the CPU cost grows quadratically rather than linearly with the number of dimensions.

Some new clustering methods are designed for high-dimensional datasets, such as the BIRCH [82], the CLARANS [2], the CURE and the CLIQUE [4] method. The BIRCH method uses a hierarchical data structure called *CF-tree* to incrementally build clusters. The CURE method uses more than one representative point for each cluster and it adjusts well to different shapes of clusters. The CLARANS method uses a restricted search space to improve the efficiency, while the CLIQUE method tends to discover clusters in all subspaces of the original data space. These clustering

methods are much more complicated than the k-means methods. However, like a regular k-means method, they do not help to identify locally correlated clusters along arbitrary directions.

3.2.2 LDR

A technique called *Local Dimensionality Reduction*(LDR) is proposed in [22]. It tries to find local correlations in a dataset and performs dimensionality reduction on the locally correlated clusters individually. The partitioning of the data and the dimensionality reduction are carried out simultaneously.

MaxReconDist is a parameter specified by users to restrict the amount of information loss within cluster. A point P in a cluster must satisfy $ReconDist(P, S) \leq MaxReconDist$ where $ReconDist(P, S)$ of a point P from a cluster S measures the distance between the originally dimensional representation of P and its approximate representation in the reduced-dimensional subspace S. The higher the error, the more the distance information lost. The LDR method also restricts the maximum number of dimensions ($MaxDim$) and minimum number of points in a cluster for indexing issue.

The clustering algorithm starts with spherical clusters. Then PCA is performed on each cluster individually. Finally, points are "reclustered" based on the correlation information to obtain the correlated clusters. A point is reassigned to a cluster that requires the minimum subspace dimensionality to satisfy the reconstruction distance bound $ReconDist(P, S) \leq MaxReconDist$, i.e., for each point, for each cluster, the minimum number of dimensions required to approximate the point (with error at most equal to $MaxReconDist$) is determined. And the cluster requiring the minimum number of dimensions N_{min} is determined. If $N_{min} \leq MaxDim$, the point is added to that cluster, and the required number of dimensions is recorded. If there is no such cluster, the point is added to the outlier set.

The quantities M, $threshold$, ϵ, $MaxReconDist$, $MaxDim$, $FracOutliers$, and $MinSize$ must be provided by the user.

An index structure to support point, range and k-NN queries is created on entire dataset. Clusters are indexed separately by an existing multi-dimensional indexing structure like the hybrid tree [21] and a single root node then connects them together. The index of each cluster is built on the d_i + 1-dimensional space, while d_i denotes the dimensionality of subspace in a cluster and the d_i+1-th dimension is the value of $ReconDist(P, S)$.

Algorithms for range search and k-NN search are also provided for similarity search on a LDR index. For k-NN search, it maintains a priority *queue* and performs breadth-first search throughout the trees until k-nearest neighbors are found. The distance between a query point and an inner node is computed in the reduced-dimensional space and the distance between the query point and a data point is computed in the original-dimensional space. Since the reduced-dimensional distances are lower bound the original-dimensional distance, the results should be the exactly k-nearest neighbors. Since this algorithm is utilized in this this study, it is described in Appendix A in more detail.

3.2.3 MMDR

Multi-level Mahalanobis-based Dimensionality Reduction (MMDR) was proposed in [40]. MMDR can be considered as an extension of LDR. The features that are different from LDR are: first, it argues that the locally correlated clusters are elliptical-shaped instead of spherical shaped, which can be explored by the Mahalanobis distance instead of the Euclidean distance. Second, it states that certain level of lower dimensional subspaces may contain sufficient information for correlated cluster discovery in the high-dimensional space.

An algorithm that discovers elliptical clusters using the low-dimensional subspace

is provided. *MPE* is the average *representation error* (like the reconstruction error *ReconDist* in LDR) of all points when they are mapped from the original space to the eliminated subspace. *MaxMPE* is specified by users as the maxinum MPE allowed. The algorithm consists of two major steps. The first step *Generate Ellipsoid* is as following:

1. Project data into its first s principal components (s should be very small).
2. Cluster on s-dimensional space using *elliptical k-means* algorithm.
3. For each cluster, project data into its local first s principal components, increase s to $2s$ and recursively call this procedure until $MPE \geq MaxMPE$ or original dimensionality is reached, then the cluster qualifies for the next procedure.

The above procedure produces possible ellipsoids in their s-dimensional space, and the dimensionality of each cluster can be further reduced by calling the second step: *Dimensionality Optimization*, which keeps decreasing the dimensionality by 1 until the change of MPE is less than a pre-set threshold. Another threshold value β is employed to determine whether a point belongs to a cluster. If the projection error is greater than β, the point is considered an outlier.

The MMDR use an index structure that represent high-dimensional data in a single dimensional space and index them with a B^+-tree. A detailed description of this method called *iDistance* can be found in [80].

The process of k-NN querying is as follows: Given a query point Q, finding k nearest neighbors begins with a query sphere defined by a *relatively* small radius r centered at Q. Q is projected into each cluster and then, step by step, the radius r is enlarged and at last when the distance of the k-th nearest neighbor is less than r the search stops. It describes some cases needed to consider for each step when r is enlarged, but does not give a clear description on how the k-NN search is going on and how to decide r according to k.

The LDR method uses individual error *reconDist* of each point as the criterion for dimensionality reduction, while the MMDR uses the average error of all points *MPE*. The two criteria are actually similar to the NMSE described in the previous section, but they are not as good as the NMSE for addressing the information loss due to dimensionality reduction, because they are absolute values, which are data-dependent, while the NMSE is an ratio, which is data-independent.

3.3 The CSVD Method

The CSVD is a local dimensionality reduction method that applies clustering first, followed by performing SVD for each cluster, and then do dimensionality reduction in a global manner [19]. In this section, three variations for implementing dimensionality reduction step are considered and the approximate k-NN algorithm for CSVD is described in detail.

3.3.1 CSVD Implementation Steps

The CSVD proceeds according to the following steps:

1. Studentization

Usually, a dataset should be preprocessed before clustering by appropriately scaling the numerical values of the indexed feature to equalize their relative importance for k-NN queries with the Euclidean distance [19]. It might be unnecessary for a synthetic dataset where the selected features have the same scale.

Set $x_{i,j} \leftarrow (x_{i,j} - \mu_j)/\sigma_j$, $1 \leq i \leq M$, $1 \leq j \leq N$, where μ_j and σ_j are the mean and standard deviation of the j^{th} column.

2. Selecting Dimensionality Reduction Targets

Dimensionality reduction can be carried out based on a target compression ratio, defined as the ratio of the "volume" of the original dataset $V_o = M \times N$ and the dimensionality reduced dataset plus the metadata:

$$V_{DR+} = H \times N + H \times N^2 + \sum_{h=1}^{H} m_h \times p^{(h)}. \tag{3.3}$$

The first two terms are due to the metadata: the space required for the centroids and eigenvectors V for all clusters. The last summation is the volume of all clusters, where m_h (resp. $p^{(h)}$) is the number of points (resp. number of dimensions) retained in cluster h or C_h.

The user can specify the compression ratio and compute the NMSE, but very aggressive data compression might result in a very high NMSE and unacceptable recall and precision. Since for a given compression ratio the NMSE varies widely over datasets, the preferred method is to specify a *target NMSE - TNMSE*, which is selected to be small enough to ensure a satisfactory precision and recall.

3. Clustering the Dataset

Partition X into H clusters $X^{(h)}, h = 1, ..., H$ with m_h points for cluster C_h by using the k-means method described on Section 3.2.

For each cluster C_h store the centroid $\mu^{(h)}$ and the radius $R^{(h)}$, which is defined as the distance between the centroid and the farthest point belonging to the cluster from the centroid.

The choice of an appropriate number of clusters H for clustering remains an open problem. For a given compression ratio increasing H results in a reduction in NMSE, but a point of diminishing returns is reached after a certain H [72].

4. Apply the SVD or PCA to Each Cluster

Compute the eigenvectors $V^{(h)}$ and the eigenvalues $\lambda_1^{(h)}, ..., \lambda_N^{(h)}$ for each cluster. Vectors of $X^{(h)}$ are rotated onto $Y^{(h)}$ according to

$$Y^{(h)} = (X^{(h)} - \vec{1}_{m_h} \mu^{(h)^T}) V$$

where $\vec{1}_{m_h}$ is a $m_h \times 1$ vector with all elements equal to 1. If there are too few points

in a cluster (less than a small multiple of N), the original dimensionality is kept. The code for SVD is from the *Numerical Recipes* package [63].

5. Selecting Retained Dimensions in Each Cluster

For a given TNMSE (target NMSE), the goal is to minimize the number of dimensions required to attain it. Let $p^{(h)}$ be the number of dimensions retained by the C_h and assume $\lambda_j^{(h)} (j = 1, ..., N)$ are in a decreasing order. The local NMSE for each cluster can be obtained using Equation 2.6 and the global NMSE is given as follows:

$$NMSE = \frac{\sum_{h=1}^{H} m_h \cdot \sum_{j=p^{(h)}+1}^{N} \lambda_j^{(h)}}{\sum_{h=1}^{H} m_h \cdot \sum_{j=1}^{N} \lambda_j^{(h)}} \tag{3.4}$$

Three methods are considered to perform dimensionality reduction on the clusters of a dataset:

Local Method (LM): According to Equation 3.4, LM removes dimensions, starting with the smallest eigenvalue, until $\text{NMSE}^{(h)} \approx \text{TNMSE}$ and $\text{NMSE}^{(h)} \leq \text{TNMSE}$.

Global Method 1 (GM1): Construct an array $\mathcal{L}$ with $H \times N$ elements, where the j-th element is a triple $\{\delta_j, \theta_j, \lambda_{\theta_j}^{(\delta_j)}\}$, where $\delta_j \in \{1, ..., H\}$ is the cluster number, $\theta_j \in \{1, ..., N\}$ is the dimension index, and $\lambda_{\theta_j}^{(\delta_j)}$ is the eigenvalue associated with ${\theta_j}^{th}$ dimension of cluster δ_j. Let $\lambda'_j = \lambda_{\theta_j}^{(\delta_j)}$. The elements of $\mathcal{L}$ are sorted in increasing order of eigenvalues, so that $\lambda'_{j+1} \geq \lambda'_j$, $j \in \{1, ..., H \cdot N\}$. Dimensions are removed from the top until the TNMSE is achieved. This is the method utilized in [19].

Global Method 2 (GM2): Similar to GM1 but replace the third element of the triple with $\lambda_{\theta_j}^{(\delta_j)} \cdot m_{(\theta_j)}$. The intuition behind GM2 is that clusters with more points should retain more dimensions, since they will be queried more frequently.

The LM retains more dimensions than necessary in comparison with global methods, as shown in the experiments. This is due to a bin-packing effect, which

allows a larger number of dimensions with smaller eigenvalues to be discarded in the latter cases. The global methods solve the problem by sorting all eigenvalues, although they come from different clusters, and removing dimensions according to one $TNMSE$. By using global methods, local NMSEs may not be the same, some may be larger than $TNMSE$ and others may be smaller, while the global NMSE is guaranteed to be equal to *TNMSE*.

The global methods should be better than the local method. Although CSVD captures the local structures of a dataset by clustering, there do exist some relationship among those clusters. Doing dimensionality reduction separately and independently cluster by cluster ignores the global relationship among the clusters. Furthermore, global methods result in the least error accumulation since they round to the upper bound of the retained dimensions only once.

The experiments show that *Global Method 2* is not as good as *Global Method 1*. A possible explanation could be: In equation 3.4, when the global NMSE is calculated, m_h can be considered as a weight, if sort the eigenvalue times m_h instead of the eigenvalue itself, m_h is considered twice.

Global methods may result in some clusters having no dimension left. In this case no index is built for the specific clusters and instead, the original data are scanned to find the k nearest neighbors. Conversely, if all the N dimensions are retained, building an index might in fact degrade performance. Rather than imposing a restriction such as $MaxDim$ in [22], a high dimensionality index is built and its performance is measured, although from a practical viewpoint a sequential scan would have been more efficient.

6. Constructing the Within-Cluster Indices

Due to the clustering and dimensionality reduction, the size and dimensionality of each cluster are much smaller than the original dataset. Therefore they are much more amenable to efficient indexing than the entire dataset. For the experiments

sequential scan is used to show the performance improvement obtained purely by CSVD. In an earlier study the authors considered the *ordered partition index* [45], which is main memory resident.

3.3.2 Approximate k-NN Search Algorithm

CSVD supports approximate k-NN queries, which can yield *false alarms* and *false dismissals* [27]. The distance metric used in this section is Euclidean distance. In order to achieve a certain accuracy $k^* \geq k$ points are retrieved.

The following steps are followed:

1. **Preprocess the Query Vector:** Studentize the query point Q as the other points in the dataset.

2. **Identify the Primary Cluster:** The *primary cluster* is the cluster to which Q belongs. In the case of k-means clustering method it is simply the cluster with the closest centroid to Q. Otherwise the cluster encoding method is used to determine the primary cluster.

3. **Compute Distance from Clusters:** The distance of a query point Q from a cluster C_h is defined as its distance from the centroid of the cluster ($\mu^{(h)}$) minus its radius. This distance is zero if the point is within the radius of C_h.

$$D(Q, C_h) = \max\{[D(Q, \mu^{(h)}) - R^{(h)}], 0\}$$

   Clusters are sorted in increasing order of distance, with the primary cluster in first position.

4. **Search the Primary Cluster:** This step produces a list of k^* points sorted in increasing order of distance. Let r_{max} be the distance of the farthest point from the query point Q in the list.

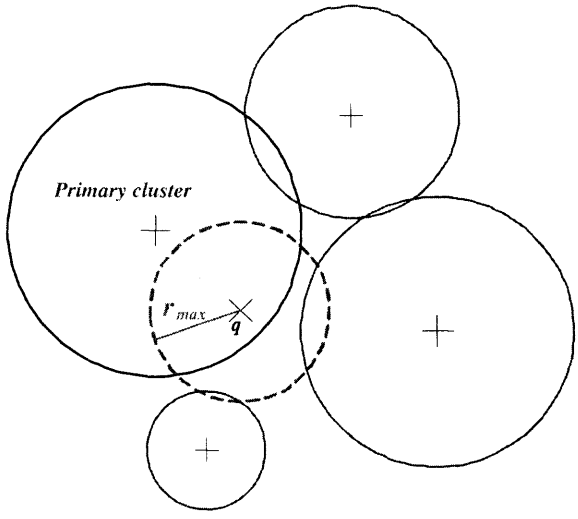


Figure 3.3 Searching nearest neighbors across multiple clusters.

5. **Search the Other Clusters:** Search the other clusters in the order obtained by Section 3. A cluster C_h is to be searched if $D(Q, C_h) \leq r_{max}$ or if the number of results is less than k^*, otherwise the search terminates (Figure 3.3). Since r_{max} is updated after a cluster is searched, the potential number of clusters to be visited is trimmed as going along.

6. **Merge the Search Results:** Distinguish between two approaches for merging k-NN results obtained from the different clusters:

   On-the-fly merging: While searching clusters, if points closer than the farthest neighbor from the query point are found, they are inserted into the list of current results dynamically, and r_{max} is updated.

   In this approach, it is needed to compare the distances among multiple clusters. Since each cluster has different number of retained dimensions, care must be taken when performing the within-cluster search. Since the data points stored in a cluster are in reduced-dimensional space, one has to use the reduced distances between the projections of Q and points in a cluster instead of original-dimensional distances. Simple geometry shows

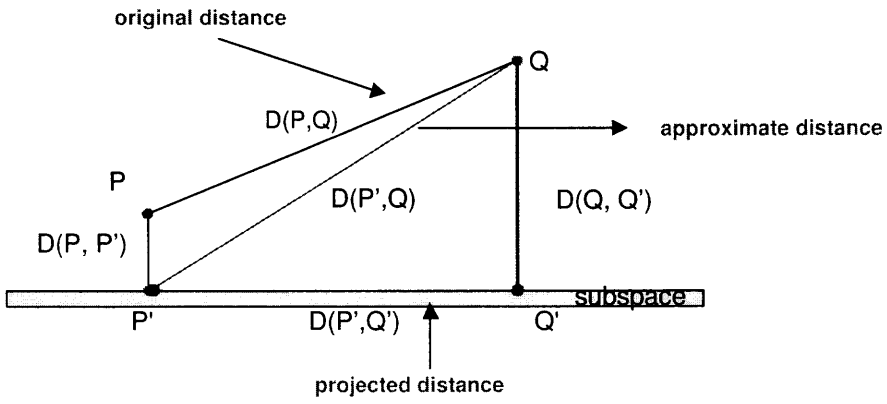


Figure 3.4 Approximate distance function.

that two points that are arbitrarily far apart in the original space can have arbitrarily close projections. The search algorithm must therefore account for this approximation by relying on geometric properties of the space and of the index construction method. As shown in Figure 3.4, CSVD approximates the squared distance $D(Q, P)$ between Q and data points P with $D(Q, P')$, where P' is the projection of P onto the cluster subspace. $D(Q, P')$ *approximate distance* is called and the distance between P' and Q' is denoted as *projected distance.* Hence

$$D^2(Q, P) \approx D^2(Q, P') = D^2(Q, Q') + D^2(P', Q') \tag{3.5}$$

Deferred-merging: A separately sorted list of k^* NNs is kept for each cluster and merge them into one list with k^* element by n-way merge sort. Here there are two choices for the distance functions:

- Use the approximate distance ($D(Q, P')$) described in Figure 3.5,
- Use the projected distance ($D(P', Q')$) for within-cluster search.

In case the distances are based on projected distances ($D(Q', P')$), in order to reconcile the distance of Q with respect to different clusters the original dataset is accessed to obtain the original coordinates for P and compute

$D(Q, P)$ before merging. Approximate distances ($D(Q, P')$) can be alternatively used for merging.

On-the-fly merging is fast and straightforward and is best suited for sequential scans of main memory resident datasets. Deferred merging is appropriate when processing k-NN queries on indexing structures, which yield the k-nearest neighbors in a batch.

7. **Post-processing:** The distance between the query point Q and the k^* resulting points are computed in the original space and the closest k results are returned. This may already have been accomplished in **Step 6**.

3.4 Performance Study

3.4.1 Experimental Setup

As summarized in Table 3.1, three real-world datasets (TXT55, AERIAL56, and COLH64) and one synthetic dataset (SYNT64) are used in the experiments. SYNT64 and COLH64 are also the datasets used to investigate the performance of the LDR method. TXT55 and AERIAL56 are studentized but COLH64 and SYNT64 are not, since the values in different dimension have the same scale.

For the elliptical k-means algorithm, the clusters created by regular k-means is used as the input. Instead of SSQ in Equation 3.2, *MaxOutLoop* is specified by users to control the number of the iterations in which covariance matrices are recalculated. *MaxOutLoop* is set to 10 in this experiment.

The CPU costs and precisions plotted in each figure are summation over one thousand *biased* k-NN queries with $k = 20$. *Biased* means that the query points were randomly selected from the original dataset.

The experiments were carried out on a Dell Workstation (Intel Pentium 4 CPU, 2.0 GHz, and 512 MB RAM) with Windows 2000 Professional.

Table 3.1 Datasets

Name	N	M	Description/Source
TXT55	55	79,814	Gabor, spatial, and wavelet features, from 400 photos. Obtained from [19].
AERIAL56	56	149,100	Feature vectors extracted from large aerial photos. From http://vision.ece.ucsb.edu/datasets/index.html.
SYNT64	64	99,984	Synthetic dataset which is generated to have 5 local correlated clusters in subspaces of different orientations by using the algorithm from [22].
COLHIST64	64	68,014	8×8 color histograms extracted from 68,014 color images obtained from http://kdd.ics.uci.edu/databases/CorelFeatures.

3.4.2 Performance Metrics

The performance comparisons are carried out from the following viewpoints:

Compression Ratio is the space required by the original dataset X divided by the space required by the dimensionality reduced dataset (without or with the space required for metadata as in Equation 3.3 in Section 3).

Recall and Precision are two metrics used to estimate retrieval efficiency, since SVD is a lossy compression method. In k-NN search, to account for the approximation, $k^* \geq k$ is retrieved, k' are among the k desired nearest neighbors. It is easy to see that $k' \leq k \leq k^*$. *Recall*, a measure of retrieval accuracy, is defined as $\mathcal{R} = k'/k$. *Precision*, a measure of retrieval efficiency, is defined as $\mathcal{P} = k'/k^*$. For a target $\mathcal{R}$, k^* starts at $k^* = k$ and is increased until this target is met and measure $\mathcal{P}$ at this point. When a fixed k^* is used (or $k^* = k$) then $\mathcal{R} = \mathcal{P} = k'/k$.

Retrieval Speedup is the ratio of the CPU time in processing k-NN queries in two different ways, which correspond to a sequential scan of the original dataset versus

(i) querying CSVD generated data using sequential scan;

(ii) querying the indexing structure described in [45].

There is a three-fold speedup because:

(a) only a subset of the clusters are searched;

(b) the dimensionality is reduced;

(c) only a subset of points in each cluster are searched due to indexing.

Possible optimizations uses the squared distance in comparisons and stops the iteration for summation when the squared distance exceeds the distance from the query point to the farthest nearest neighbor (r_{max}).

With the first two terms pre-computed, the inner product in $D_2(P,Q) = \| \vec{p} \|_2 + \| \vec{q} \|_2 - 2\vec{p}.\vec{q}$ can be computed very efficiently using IBM's ESSL package for PowerPC [19], but not on an *X86* processor.

3.4.3 Experiments

In this chapter, the optimized sequential scan, which bypasses any unnecessary distance calculations, is used for within-cluster search instead of any multi-dimensional indexing structures. Thus the relative performance of SVD, CSVD, LDR and MMDR can be studied.

The first experiment is to compare the performance of CSVD with that of the global SVD, the second experiment is to compare CSVD with LDR, and the third one is to compare CSVD with MMDR.

I. Comparison of CSVD and SVD

The following experiments are carried out to illustrate the different facets of CSVD.

Compression Ratio: First compare SVD with CSVD to determine the improvement in the compression ratio. TXT55 is partitioned into varying number of clusters and the average number of dimensions retained and data volume (Equation 3.3) are shown in Figure 3.5. It is clear that CSVD reduces more dimensions than SVD for a given NMSE. Furthermore, it follows from Figure 3.5(a) that CSVD reduces more dimensions as the number of clusters increases. However, a point of diminishing returns is soon reached and for a very large number of clusters, e.g., 128 clusters in Figure 3.5(b), the index volume exceeds that of 32 and 64 clusters. This is due to the increased space required for metadata (see Equation(3.3) in Section 3.3). According to this experiment, 32 or 64 clusters can be the final choice for H.

The effect of the different options of CSVD is studied as discussed in Section 3. LM is outperformed by GM1 and GM2 (Figures 3.5(c) and 3.5(d)). It follows from Equation 3.4, which takes into account the number of points in each cluster, that using $m_h\lambda^{(h)}$ in GM2 is an overkill. Consequently, for all datasets in this paper GM1 outperforms GM2.

Also the compression ratio of the regular k-means and the elliptical k-means is compared using GM1 on the TXT55 dataset which creates 16 clusters. The results are shown in Figure 3.6. From 3.6 (a) it is easy to see that with same level of the NMSE, the elliptical k-means is always able to retain less dimensions than the regular k-means. Figure 3.6 (b) shows that as the NMSE increases, the advantage of the elliptical k-means over the regular k-means becomes more significant.

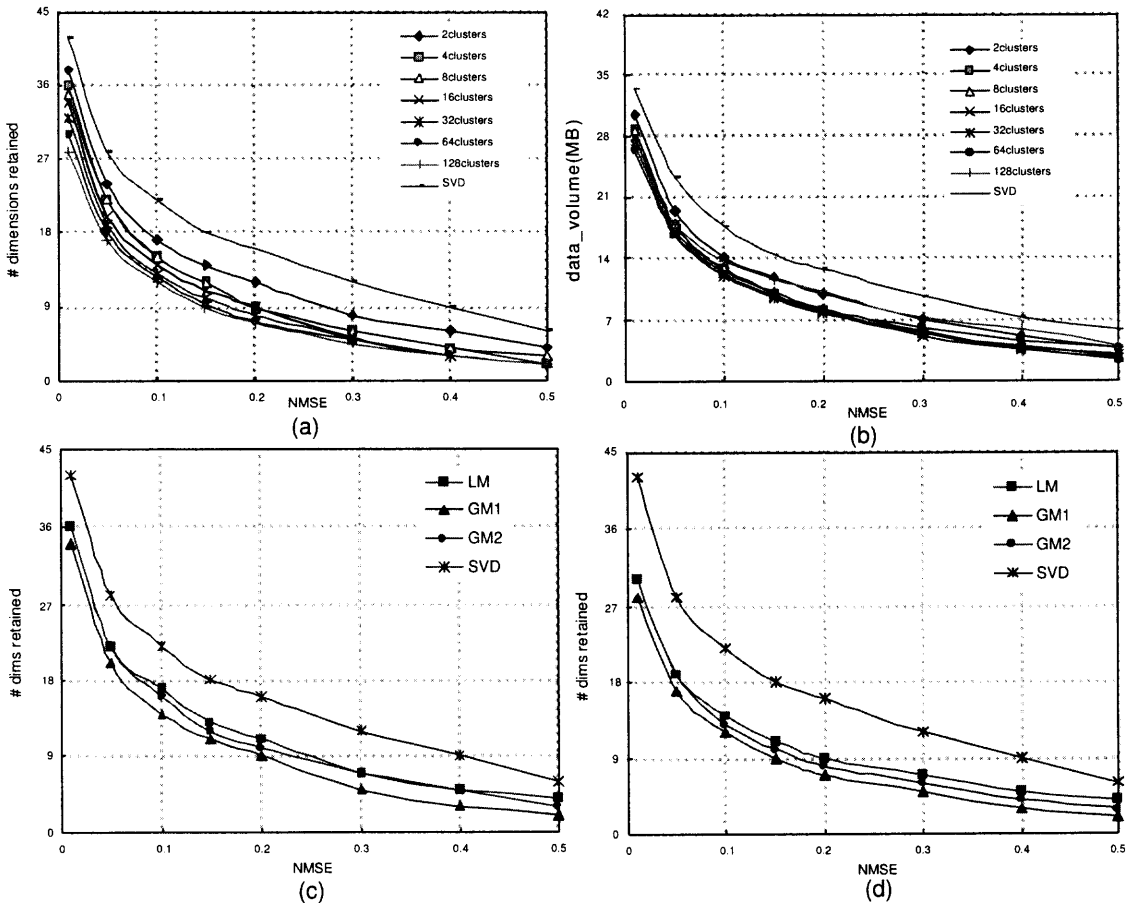


Figure 3.5 (a) The average number of dimensions retained by SVD and CSVD-GM1 for different number of clusters vs. NMSE. (b) Data volume of SVD and CSVD-GM1 vs. NMSE. (c) Number of dimensions retained by the three CSVD methods for 16 clusters vs. NMSE. (d) Same with 128 clusters. TXT55 was used in all cases.

Retrieval Speedup: One reason for the speedup is that data clustering and the approximate k-NN querying reduce the number of data points visited as well as number of dimensions checked during query processing. CSVD prunes the search space by visiting a small number of clusters (Figure 3.7(c)), so that only a small fraction of points is visited (Figure 3.7(a)).

Figure 3.7(b) provides the speedup, obtained by SVD and CSVD methods versus NMSE with different degrees of clustering, with respect to sequential scan for SYNT64. It is observed that over a 30-fold speedup is obtained for $H = 10$ for

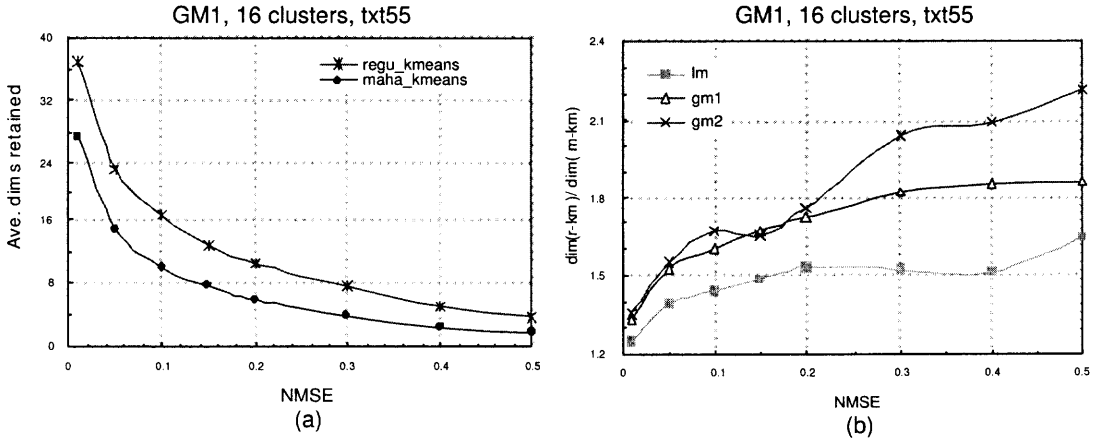


Figure 3.6 (a) Number of dimensions retained for regular k-means and elliptical k-means using GM2 as a function of NMSE. (b) Ratios of number of dimensions retained by regular k-means to elliptical k-means for three methods.

smaller NMSEs.

Figure 3.7(d) shows the speedup obtained by SVD and variants of CSVD, while maintaining $\mathcal{R} = 0.75$, for AERIAL56. It is observed that CSVD variations provide a higher speedup than SVD and that GM1 is the best.

The CPU cost of regular k-means and elliptical k-means can be found in Appendix B.

Quality of Retrieval: Figure 3.8 shows the precision versus the NMSE for the three CSVD methods and SVD with $\mathcal{R} = 0.75$. For both datasets, the results of CSVD are better than SVD and among the three CSVD methods, GM1 is the best.

On-the-fly versus deferred merging: Figure 3.9 compares on-the-fly merging with deferred merging from the viewpoint of CPU cost and precision. In Figure 3.9(a) for 128 clusters, when $NMSE = 0.4$, the CPU cost of deferred merging is twice as high as that of on-the-fly merging. This is because deferred merging requires access to the original dataset residing in the main memory. On the other hand,

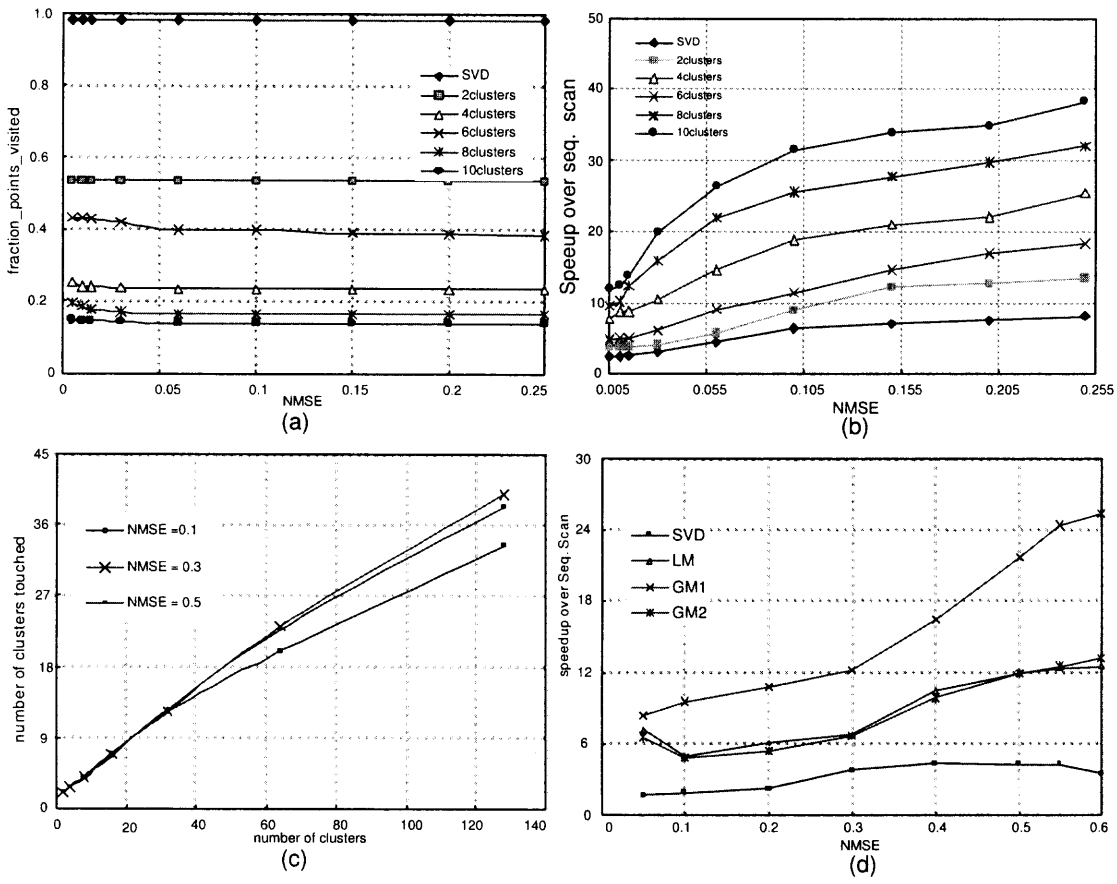


Figure 3.7 (a) The fraction of data points visited ($k^* = k$, SYNT64). (b) Speedup of CSVD and SVD vs. NMSE ($k^* = k$, SYNT64). (c) The average number of clusters touched during the search (R = 0.75, AERIAL56). (d) Speedup of the three CSVD methods and SVD (32 clusters, R = 0.75, AERIAL56). CSVD-GM1 utilized in (a), (b) and (c).

deferred merging outperforms on-the-fly merging in precision (Figure 3.9(b)). This is due to the fact that deferred merging visits the original dataset and obtains the original distances between points.

Accessing or Not Accessing Database: Two conditions under different distance functions are considered. **Accessing database:** Within-cluster-search uses projected distance to find k candidates for each cluster, then locates them in the database and calculate the original distance between the candidates and

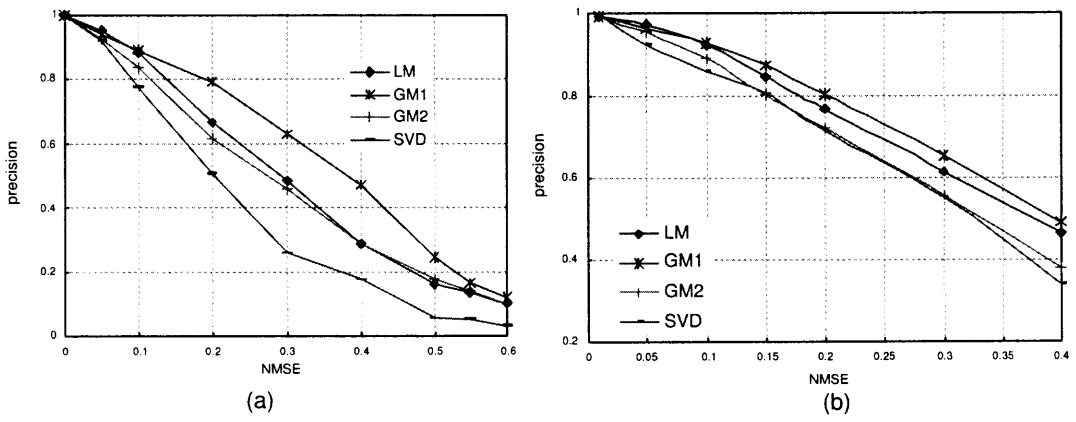


Figure 3.8 Precision vs. NMSE for the three CSVD methods and SVD for (a) 16 clusters, AERIAL56, R = 0.75 and (b) 128 clusters, TXT55, R = 0.75.

query points before merging them to a final list of k points. Since k and H (the number of clusters) are both small constants, this will not increase the I/O cost much. **Not-accessing database:** Within-cluster-search uses approximate distance to find k candidates for each cluster and uses the same distance to merge.

Figure 3.10 illustrates the curves of CPU cost (3.10(a)) and precision (3.10(b)) for *deferred-merging* using the original distance and approximate distance, as well as *on-the-fly-merging* using approximate distance. It turns out that they have similar precisions, while *deferred-merging* using approximate distance is a compromise between the other two cases.

II. Comparison of CSVD and LDR

For CSVD, the best of the three proposed methods, which is GM1, is used. For LDR, even for the same set of parameters, the results differ from one instantiation to another because of the randomized choice of centroids for spatial clusters. Therefore for each set of parameters LDR is run over 10 times and the best configuration which results in the smallest value of NMSE, is selected.

Many parameters need to be specified for LDR. In order to simplify the problem,

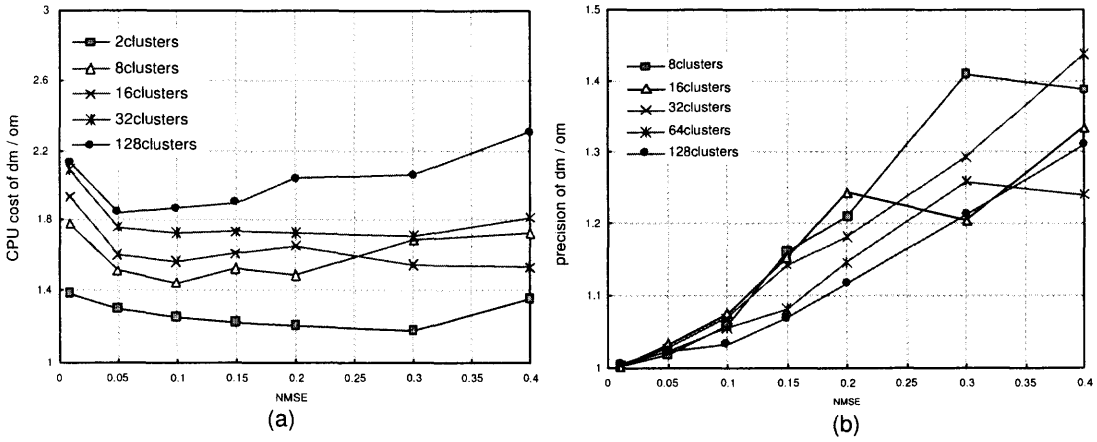


Figure 3.9 Ratios of (a) CPU cost and (b) precision of deferred merging (dm) to on-the-fly merging (om) vs. NMSE. CSVD-GM1 (R = 0.75, TXT55).

the *MaxDim* (maximum subspace dimensionality of a cluster) is set to the original dimensionality, and *MaxReconDist* (maximum reconstruction distance that a point in a cluster can have) as well as *FracOutliers* (permissible fraction of outliers) are varied to obtain different levels of approximation. NMSE, the control variable used in CSVD, is a better criterion for comparison, because it reflects the fraction of dataset variance lost and is independent of dataset characteristics.

Experiments with the Synthetic Dataset. Data points are generated using the synthetic dataset generator which was presented with the LDR method [22] to form groups that are separated from each other and with different intrinsic dimensionality. Therefore, both LDR and CSVD can generate good clusters with little overlap. This can be seen from the fraction of data points which has to be visited during query processing (Figure 3.11(d)). It can be seen that CSVD has a better compression ratio than LDR on the average (Figure 3.11(a)), and the precision of k-NN queries for LDR and CSVD are similar (Figure 3.11(b)). Furthermore, CSVD has a lower CPU cost (Figure 3.11(c)), since it visits a smaller fraction of data points.

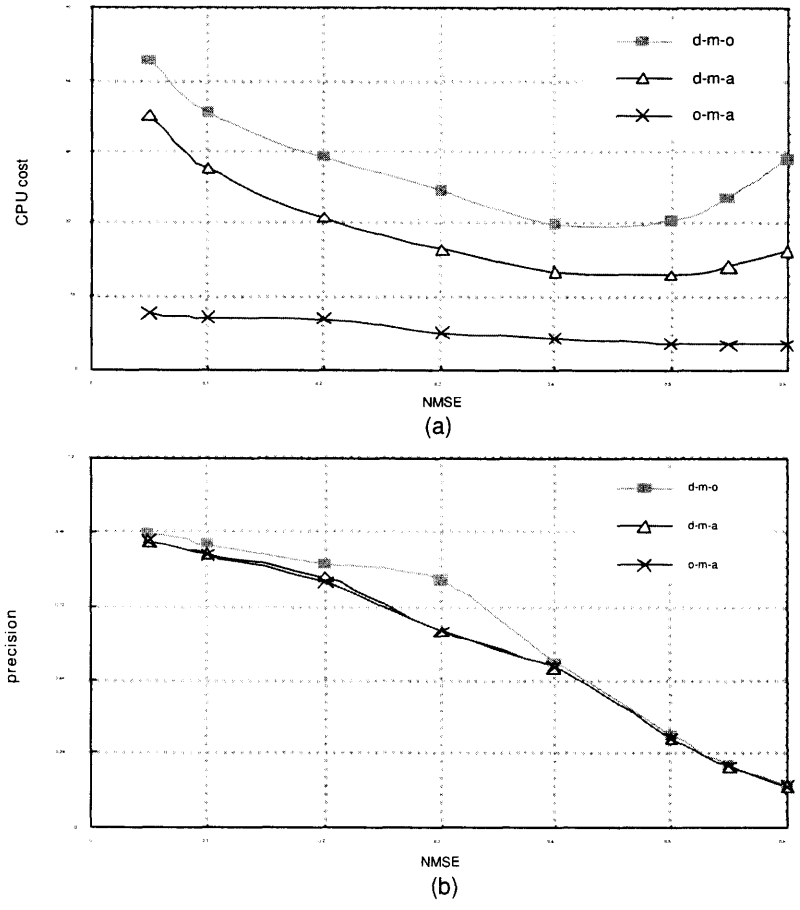


Figure 3.10 (a) CPU cost and (b) precision vs. NMSE, parameterized by deferred-merging-original distance (d-m-o), deferred-merging-approx. distance (d-m-a) and on-the-fly-merging-approx. distance (o-m-a). AERIAL56, recall = 0.8, 64 clusters, GM1.

Experiments with real-world Datasets. For high-dimensional real-world datasets considered in the experiments, both LDR and CSVD generate clusters overlap heavily, therefore a large fraction of clusters needs to be visited during k-NN querying.

From Figures 3.12, there is no significant difference between the compression ratios of LDR and CSVD for COLHIST64. CSVD outperforms LDR in precision when $NMSE < 0.20$. According to Figure 3.13 and 3.14, for TXT55 and AERIAL56, CSVD has higher precision and lower CPU cost than LDR.

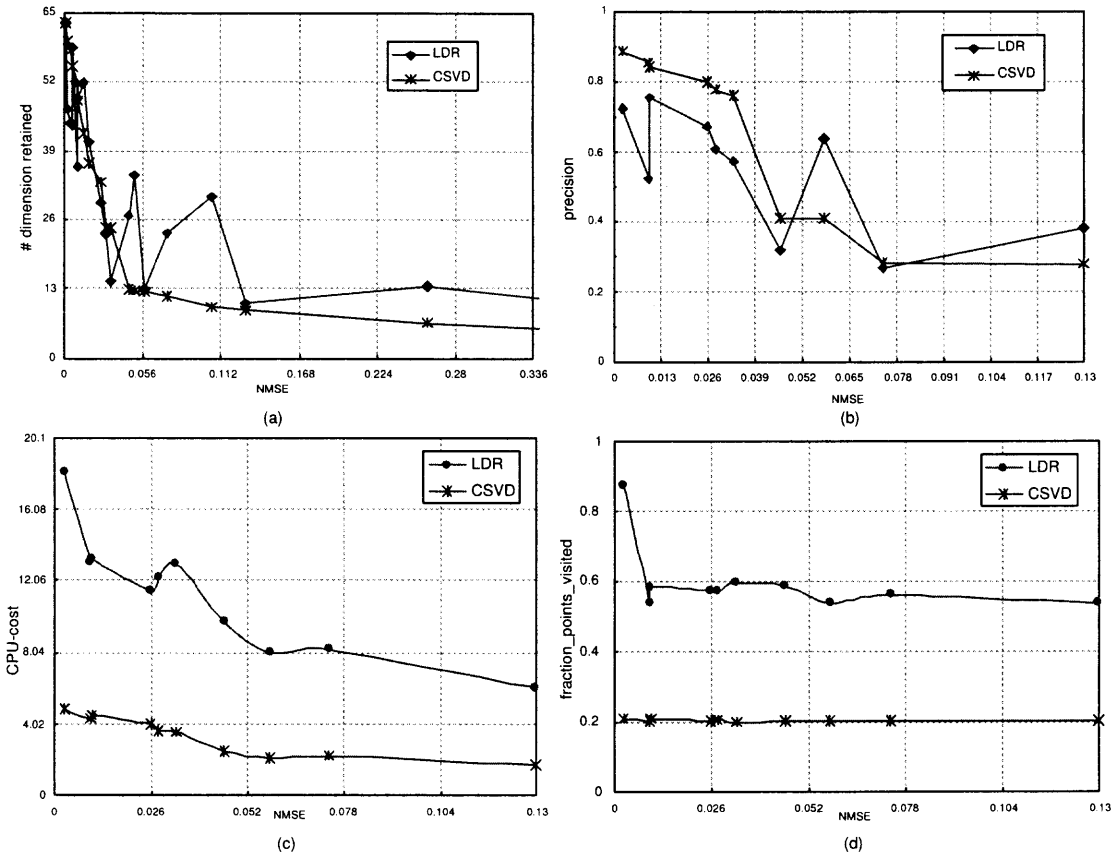


Figure 3.11 Results for CSVD(GM1) and LDR for SYNT64 (a) Average number of dimensions retained (b) Precision vs. NMSE. (c) CPU cost vs. NMSE. (d) Fraction of data points visited vs. NMSE.

Summary of the Comparison. Neither method outperforms the other in all cases. The two methods yield similar compression ratios. Since CSVD partitions datasets by a regular k-means method and reduces the dimensionality in a global manner, the whole process is conceptually much simpler than LDR. CSVD is a more adaptive method because clustering and dimensionality reduction are carried out independently. In most cases CSVD outperforms LDR as far as retrieval efficiency and CPU cost are concerned.

On the other hand, LDR has the potential to better identify more "SVD friendly" clusters [72] and hence outperforms CSVD in other cases, e.g., when a

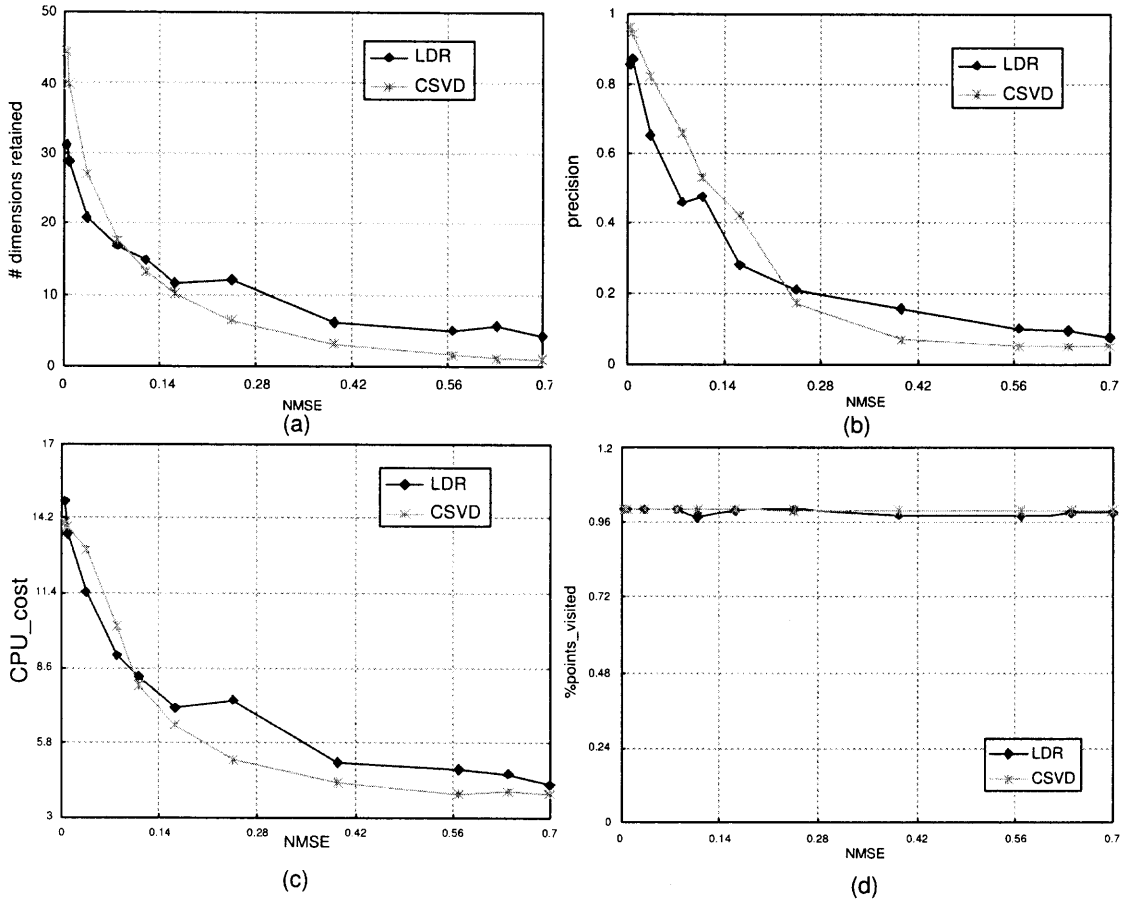


Figure 3.12 Results for CSVD(GM1) and LDR for COLHIST64 (a) Average number of dimensions retained (b) Precision vs. NMSE. (c) CPU cost vs. NMSE. (d) Fraction of data points visited vs. NMSE.

dataset has very clear local correlation. It might yield a better compression ratio and retrieval efficiency if its parameters are chosen carefully after an exploratory analysis of the dataset. In the experiments LDR did not capture the local data structure well for studentized datasets. For a high dimensional dataset, after studentization, it is even more difficult to form good clusters, since data points are brought closer to each other. In this condition, local correlation becomes unclear. This is the reason that LDR performs not as well as CSVD for TXT55 and AERIAL56.

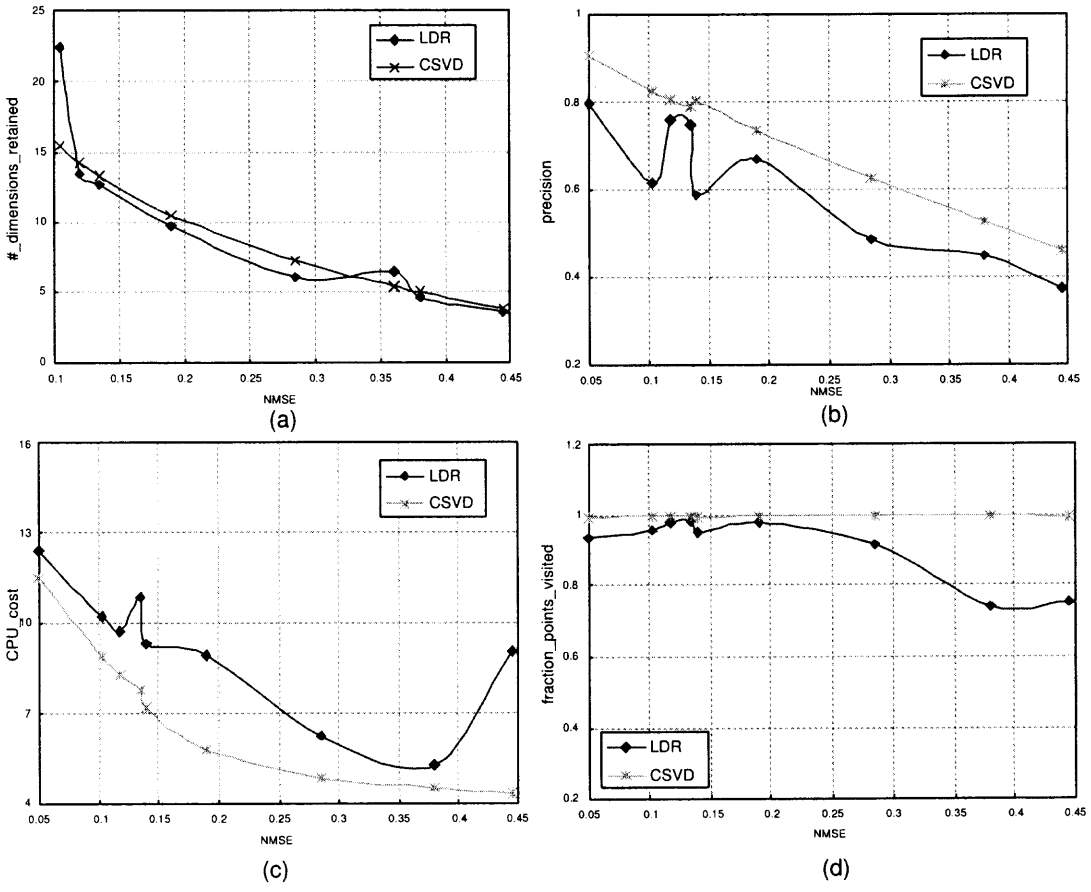


Figure 3.13 Results for CSVD(GM1) vs. LDR for TXT55 (a) Average number of dimensions retained (b) Precision (c) CPU cost (d) Fraction of data points visited.

Due to space limitations not all results are included in this paper. The reader is referred to [73] for additional experimental results.

In conclusion, CSVD seems to be the preferred method for high dimensional dimensionality reduction. It is more flexible in that it can use any clustering method according to the application and datasets.

II. Comparison of CSVD and MMDR

Experiment on the MMDR is held on only the synthetic dataset SYNT64. Firstly, several clustered datasets of the MMDR were generated by varying *MaxMPE*. Then NMSEs were calculated while CSVD clustered datasets with same values of

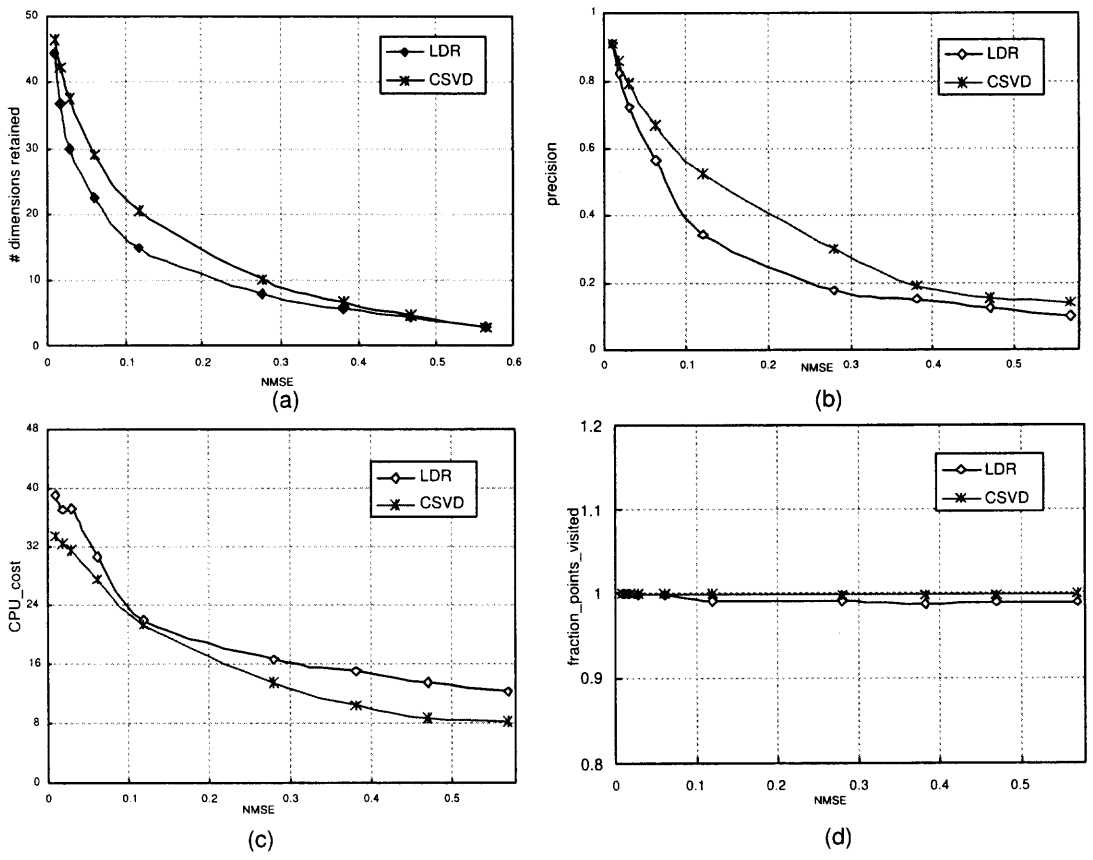


Figure 3.14 Results for CSVD (GM1) vs. LDR for AR56 (a) Average number of dimensions retained (b) Precision (c) CPU cost (d) Fraction of data points visited.

NMSEs were generated so that the CSVD and the MMDR can be compared based on the same values of NMSEs.

From Table 3.2, it is easy to see that as the MaxMPE increases, NMSE increases as well. And like LDR, when parameters change, the number of clusters might change as well. For a same value of NMSE, CSVD retains a slightly fewer dimensions than MMDR, which means the compression ratio of CSVD is higher.

As for query cost, from Figure 3.15 (a), MMDR has a slightly lower CPU cost compared to CSVD at all NMSEs. This is because CSVD retrieved a slightly more points so that the fraction of total points visited are higher than that of MMDR.

According to the above results, CSVD is a simpler method which does not

Table 3.2 MMDR Parameters and Retained Dimensions Compared to CSVD

	MMDR			CSVD
NMSE	MaxMPE	Number of Clusters	Dim Retained	Dim Retained
0	0	55	64	64
0.017	0.1	55	48.99	47.52
0.068	0.2	55	19.42	15.49
0.138	0.3	49	13.37	12.94
0.2	0.4	37	10.48	9.52

perform worse than the much more complicated MMDR method.

3.5 Conclusions

In this chapter, first some variations of CSVD that were not explored in [19] are specified and investigated. The LM dimensionality reduction, which is tantamount to applying SVD with a target NMSE (TNMSE) to individual clusters, and GM2 method, are outperformed by the GM1 method, which selects the principal components to be retained on a global basis. Also, the tradeoff in utilizing approximate distances against the original distance is investigated in processing k-NN queries.

The CSVD is compared with LDR with three real-world datasets and one synthetic dataset from the viewpoint of the compression ratio, retrieval efficiency for k-NN queries, and CPU cost for sequential scan. CSVD compares favorably or outperforms LDR in most cases.

An advantage of CSVD with respect to LDR is its flexibility in that the clustering phase is decoupled from dimensionality reduction. Very large datasets can be clustered using clustering methods applicable to disk resident data, coupled with PCA applied to the covariance matrix C (C can be computed in a single dataset scan [46]). This remains an area of future investigation.

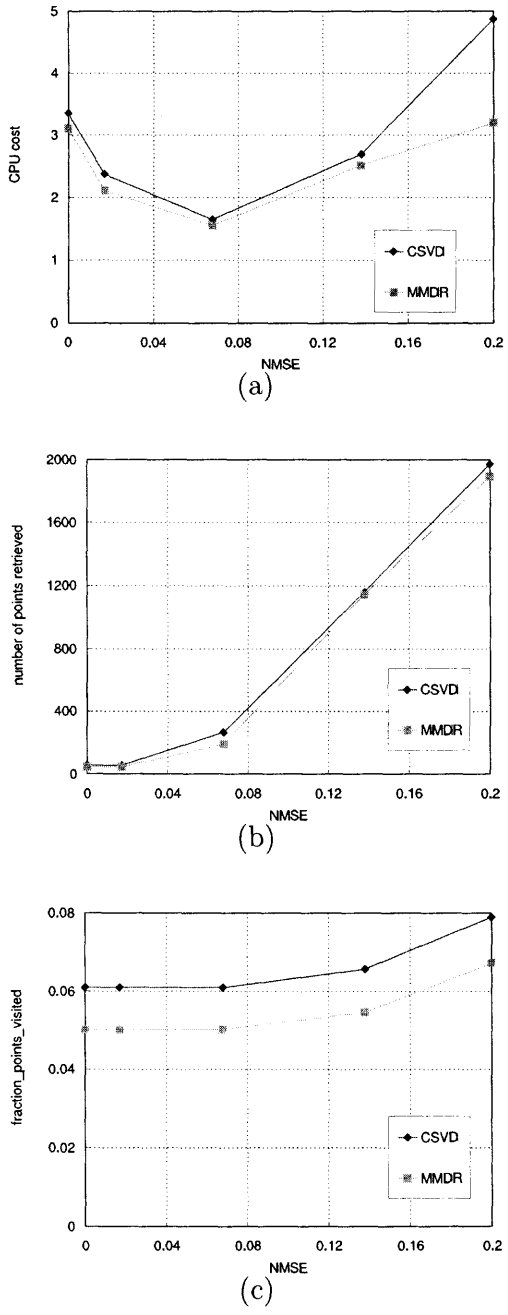


Figure 3.15 Comparison of CSVD and MMDR on (a) CPU cost (b) number of points retrieved (c) fraction of points visited. SYTH64, 20-NN.

CHAPTER 4

K-NEAREST NEIGHBOR SEARCH

4.1 Introduction

In recent years, the k-NN query has become an important tool in content-based retrieval or similarity search in multimedia databases containing images, audio and video clips, etc..

k-NN queries retrieve k closest objects to the query object. Multimedia objects are usually represented by features, such as color, shape and texture. Features are then transformed into high-dimensional points (vectors). Similarity queries, especially k-NN queries based on a sequential scan of large files or tables of feature vectors is computationally expensive and result in a long response time. Multi-dimensional indexing methods fail to work efficiently in high-dimensional space due to the problem of the dimensionality curse.

A well-known technique is to reduce the dimensionality of feature vectors and then build a multi-dimensional index structure on the dimensionality reduced space. Techniques based on linear transformations, like the SVD, have been widely used for this purpose.

Clustering and Singular Value Decomposition (CSVD) is proposed in [19] to solve the problem by clustering the dataset, reducing the dimensionality and building index in the dimensionality reduced space for each cluster. It has been shown to outperform global SVD when the data are locally correlated.

An algorithm to find k nearest neighbors has been proposed especially for CSVD (See Section 3). It is an approximate method since it violates the lower-bounding property [27]. The lower-bounding property which is initially defined for range queries also works for k-NN queries, since a k-NN query can be transferred to a range query

with an estimated search radius [48].

In this chapter, an exact k-NN algorithm is presented for CSVD based on a multi-step k-NN search algorithm presented in [48] which is designed for global dimensionality reduction methods. Experiments with two datasets show that it requires less CPU time than the approximate algorithm at a comparable level of accuracy.

The rest of the chapter is organized as follows: Section 4.2 gives the related work. Section 4.3 describes the new algorithm in details. The experimental results are given in Section 4.4 and the conclusion appears in Section 4.5.

4.2 Related Work

There are two categories of nearest neighbor search methods: *exact methods* ([66, 37, 48, 68]) and *approximate methods* ([31, 7, 19]). Exact k-NN algorithms retrieve the k points which are exactly the same as those obtained from original space using sequential scan. For datasets with extremely large number of points and very high dimensionalities, it is expensive to obtain exact results. Furthermore, the meaning of "exact" has been questioned [15] because using feature vectors and a distance function to address similarity of multimedia object itself is just a heuristic [1]. In this case, approximate methods may be more efficient at the cost of a lower accuracy for k-NN queries.

For querying indexes built on dimensionality reduced space, if the distance between any two projected points, i.e., dimensionality reduced points, lower bounds the distance between the corresponding original points, then it is possible to have an exact k-NN algorithm. The k-NN search algorithm for CSVD in [19] is an approximate method because it uses *approximate distance* between two projected points. Approximate distance neither lower bounds nor upper bounds the original distance. Figure 4.1 shows an example in 2-dimensional space. For a data point P

and query point Q, the approximate distance is defined as

$$D(Q, P') = \sqrt{D^2(Q, Q') + D^2(Q', P')}$$

where P' and Q' are the projected points of P and Q into the subspace respectively and $D()$ is the Euclidean distance. Given that R is another data point, it is easy to see that for point P, approximate distance $D(Q, P')$ is larger than the original distance $D(Q, P)$, but for point R, $D(Q, R')$ is smaller than $D(Q, R)$.

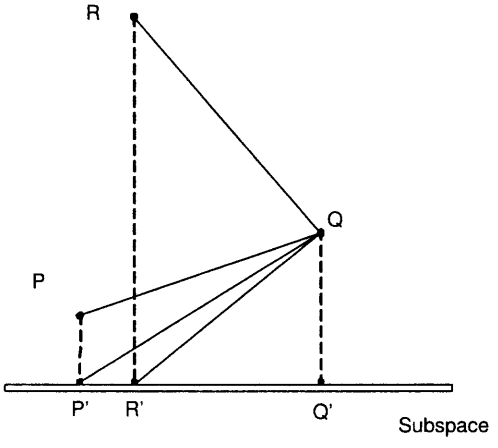


Figure 4.1 Approximate distance in CSVD.

In order to obtain a certain accuracy, $k^*(> k)$ points has to be retrieved, where the precision can not be probabilistically controlled, i.e., k^* can not be predefined as a function of *recall*. On the other hand, the good point of this method is that it doesn't need to access the original database. Exact methods, when working on dimensionality reduced indexes, have to access the original databases for post-processing.

Many algorithms for exact k-NN queries have been reported in the literature. The earliest k-NN method for multi-dimensional indexes [66] is designed for R-tree. It uses *MINDIST* and *MINMAXDIST* to prune branches. There is no dimensionality reduction involved in this basic algorithm.

The state-of-the-art of multi-step exact k-NN algorithm which can be applied on dimensionality reduced datasets or indexes appears in [48]. It has three steps:

1. Find the k nearest neighbors to the query point Q in the subspace (dimensionality reduced data space);

2. Find the actual distances of these k points to Q and especially the farthest distance (d_{max});

3. Issue a range query on the subspace with radius d_{max} and obtain their original distances to Q, then pick the closest k points as output.

The above method is independent of index structures and it can be plugged in to any existing multi-dimensional index and even sequential scan.

Another popular method called *Ranking method* [37] finds k nearest neighbors by using a priority queue and in that case k is not necessary to be a fixed number. The method introduced in [68] extends the ranking method to dimensionality reduced data and presents an exact algorithm that results in less false alarms. The algorithm for *Local Dimensionality Reduction* (LDR) [22] is also based on ranking. It uses a priority queue to navigate the index for all clusters and explores only those objects that are within the range of k-th nearest neighbor [1]. All these few methods are designed for multi-dimensional indexes and they are index-dependent.

4.3 Exact k-NN Search Algorithm for CSVD

The index structure after performing CSVD is shown in Figure 4.2. The root node contain basic data information (size, dimensionality, centroid and radius – the farthest points to the centroid) of the whole dataset – and the number of clusters as well as the address of each clusters. For each cluster, besides the basic data information, a pointer to the local index is provided. Any multidimensional index can be used as local index.

[1]This k-NN algorithm, which is also applied in Chapter 5, is described in detail in Appendix A

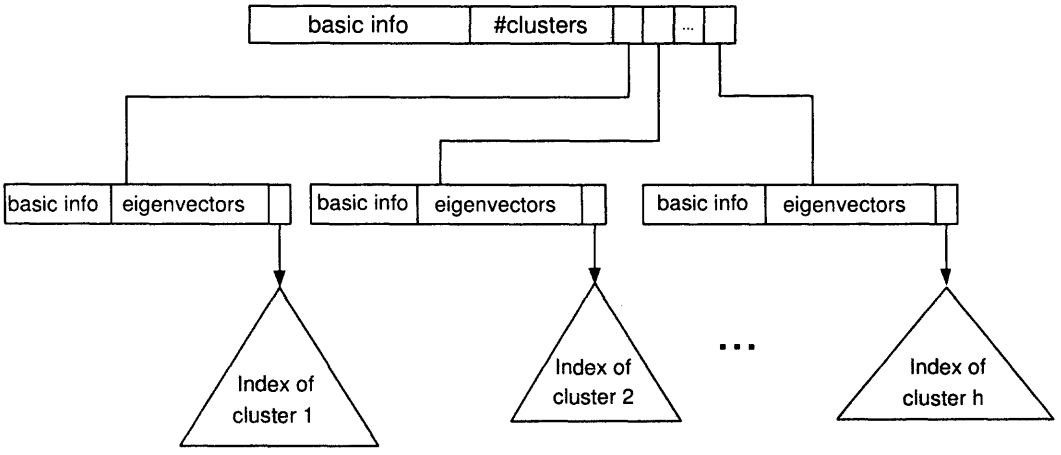


Figure 4.2 Indexing structure of CSVD.

In this chapter, to make life simple, "approximate method" just refers to the approximate k-NN algorithm for CSVD in [19], and "exact method" only denotes the newly proposed method.

Given a set of clusters $C_1, C_2, ..., C_H$ and their centroids and radius, the exact k-NN algorithm proceeds as follows, noting that the idea for pruning unnecessary clusters is actually similar to the approximate algorithm:

1. Find the primary cluster C_P, which is the cluster with the closest centroid to query Q. Order other clusters based on their distances to Q, which is defined as $dist(Q, C_h) = max(D(Q, \mu_h) - R_h, 0)$, where μ_h is the centroid of C_h and R_h is the radius of C_h.

2. Project Q onto C_P and obtain k closest points with dimensionality reduced (projected) distance.

3. Compute the original distances of the above k points to Q and return the maximum distance d_{max}.

4. Perform a range query centered at Q and with radius d_{max} in the projected space.

5. Compute the original distance of each retrieved point and insert it into the sorted list of k nearest neighbors and replace d_{max} with the distance of current k-th nearest point.

6. For the next candidate cluster C'_h, If $d_{max} > dist(Q, C'_h)$, then go to step 4 to search that cluster, otherwise stop and return the current k points in the sorted list.

Lemma 4.3.1 *The above algorithm guarantees no false dismissal for k-NN queries.*

Proof: According to Lemma 4 in [48], since the projected distance lower-bounds the original distance, Step 2, 3, 4 and 5 do not generate any false dismissal because it is just the algorithm in [48] applied to one cluster. It is necessary to prove that for clustered datasets, the same conclusion can be drawn. Consequently, Step 6 has to be proved to not generate any false dismissals.

The proof is by contradiction. Suppose $dist(Q, C'_h) = d > d_{max}$, but there is a point P' in C'_h which should be one of the k-nearest neighbors. Since d_{max} is the current k^{th} nearest neighbor, $D(Q, P') \leq d_{max} < d$. However, according to the definition of $dist(Q, C'_h)$, for all points inside C'_h, d is the smallest possible distance to Q, therefore $D(Q, P') \geq d$, which contradicts $D(Q, P') < d$. Therefore, Step 6 does not generate any false dismissal. QED.

This exact method not only guarantee 100% accuracy but also has shown experimentally to have lower CPU cost than the approximate one at a comparable level of accuracy. This is due to the following reasons:

1. In order to obtain a desired recall, the approximate method needs to estimate a value $k^* \geq k$ for k-NN queries, which can not be obtained without iteration. For the exact method, only k points needs to be retrieved.

2. For indexes which reside in main memory, *on-the-fly merging* policy [73] is utilized for collecting results of all clusters. In that case, approximate method

requires more CPU time to maintain the sorting list for the final results which could be much longer than that of the exact method.

3. For multi-dimensional indexes which resides in disks, *deferred merging* policy [73] has to be utilized for merging results of multi-clusters. Therefore, for approximate method, a k^*-NN query, which is known to be more complex than range query, is issued for each cluster. In the exact algorithm, on the other hand, k-NN query is issued only once and from then on only range queries are involved, which tend to be more efficient.

4.4 Experiments

The experiments are carried out with two datasets: TXT55 (real-world dataset with size 79,814 and dimensionality 55) and SYNT64 (synthetic dataset with size 99,972 and dimensionality 64). One thousand points are randomly selected as query points. For within-cluster search, optimized linear scan which bypasses unnecessary calculations instead of multidimensional indexes is used. The NMSE was used in [19] to illustrate the ratio of total information loss caused by dimensionality reduction. It is proportional to the index size. Here it is also used as a parameter to show the relationship between CPU cost and compression ratio of indexing.

Recall (R) and CPU cost are used to quantify the accuracy and efficiency of retrieval. Let $A(\vec{q})$ denote the k nearest points to a query point. To account for the approximation, one may retrieve a result set $B(\vec{q})$ containing $k^* \geq k$ elements. Let $C(\vec{q}) = A(\vec{q}) \cap B(\vec{q})$. Then, $R = |C(\vec{q})|/|A(\vec{q})|$.

4.4.1 CPU Cost versus NMSE for the Exact Method

First the relationship between CPU cost of the exact k-NN algorithm for different approximation level of CSVD indexing is explored, qualified by the NMSE.

In Figure 4.3, the CPU cost of k-NN queries is given versus the NMSE, for

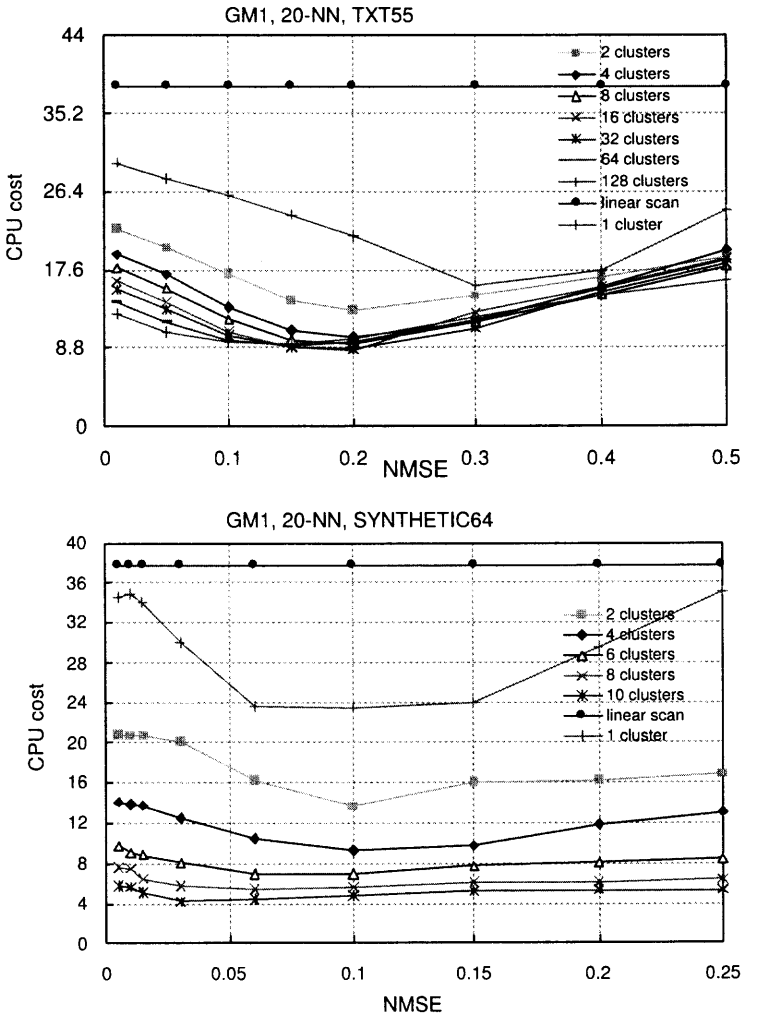


Figure 4.3 CPU cost of the exact k-NN algorithm.

different number of clusters. It can be seen that the exact method has much lower CPU cost than linear scanning the original dataset. Furthermore, as the number of clusters increases, more CPU time is saved because fewer points are visited. An interesting point is, as the NMSE becomes larger, the CPU costs exhibits a (global) minimum. This is because with fewer dimensions the index query cost decreases, while there is an increase in post-processing due to an increase of false alarms.

4.4.2 Exact Method versus Approximate Method

The next experiment is to compare the approximate k-NN algorithm with the exact algorithm at comparable recall values – for exact method, $R = 1.0$ for sure, but for approximate method $R \geq 0.995$ and $R \geq 0.9999$.

In order to obtain a given recall, approximate k-NN needs to estimate k^*. It is not easy to find the k^* without iterations, but the CPU time for finding k^* is not considered in this comparison. Here the number of clusters are chosen to be 128 for TXT55 and 6 clusters for SYNT64.

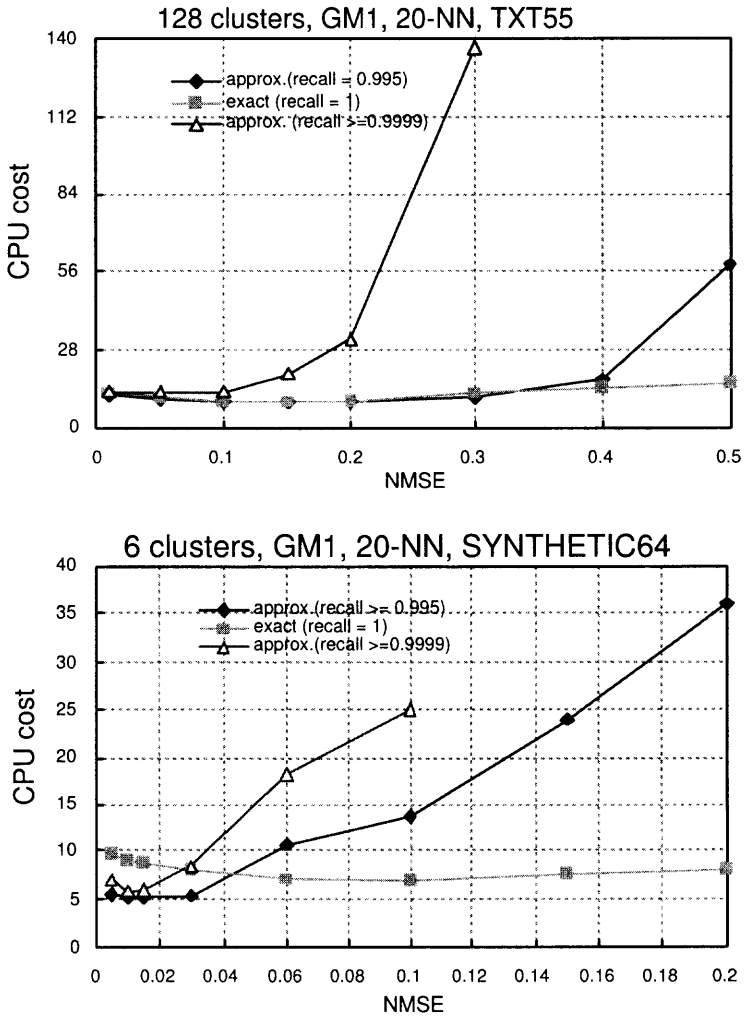


Figure 4.4 Comparison of CPU cost for the exact and approximate methods.

Figure 4.4 shows the CPU costs of the exact algorithm and the approximate algorithm, with different NMSEs. It can be seen that the exact method has much lower CPU cost than the approximate method in most cases and only under the condition that NMSE is very small (< 0.003 for SYNT64), the approximate method outperforms the exact method. It should be noticed that for the approximate method when R approaches 1.0 (e.g., $R \geq 0.9999$ for SYNT64), k^* is an unacceptable large value.

The values of k^* are given in Figure 4.5. The approximate method incurs much more CPU time when the NMSE is larger because it has to retrieve a very large number of candidates to guarantee a high recall.

4.4.3 The Effect of Maintaining an Extra Dimension

To minimize the difference between the original distance and projected distance, the method proposed in LDR paper [22] is adopted. For a given NMSE, cluster C_h keeps d dimensions after dimensionality reduction. Previously the index would be built on the d-dimensional space, here instead, $d+1$ dimensions will be kept for each data point $\vec{p} = \{p_1, p_2, ..., p_N\}$, while the extra one dimension is the *ReconDist*, defined as $\sqrt{\sum_{i=d+1}^{N} {p_i}^2}$, which is actually the lost distance information for an individual point due to dimensionality reduction. Then the Euclidean distance is computed in $d+1$-dimensional. The LDR projected distance is closer to the original distance and it is guaranteed to lower bound the original distance [22].

Figure 4.6 illustrates the relationship among the original distance, the LDR distance and the approximate distance described in Chapter 3. Note that each value of the distance with respect to a query is the sum of squared-distances of the 20 nearest neighbors to the query point. It is easy to see that roughly the approximate distance and LDR distance provide a similar level of approximation to the original distance. However, the LDR distance always lower bounds The original distance and

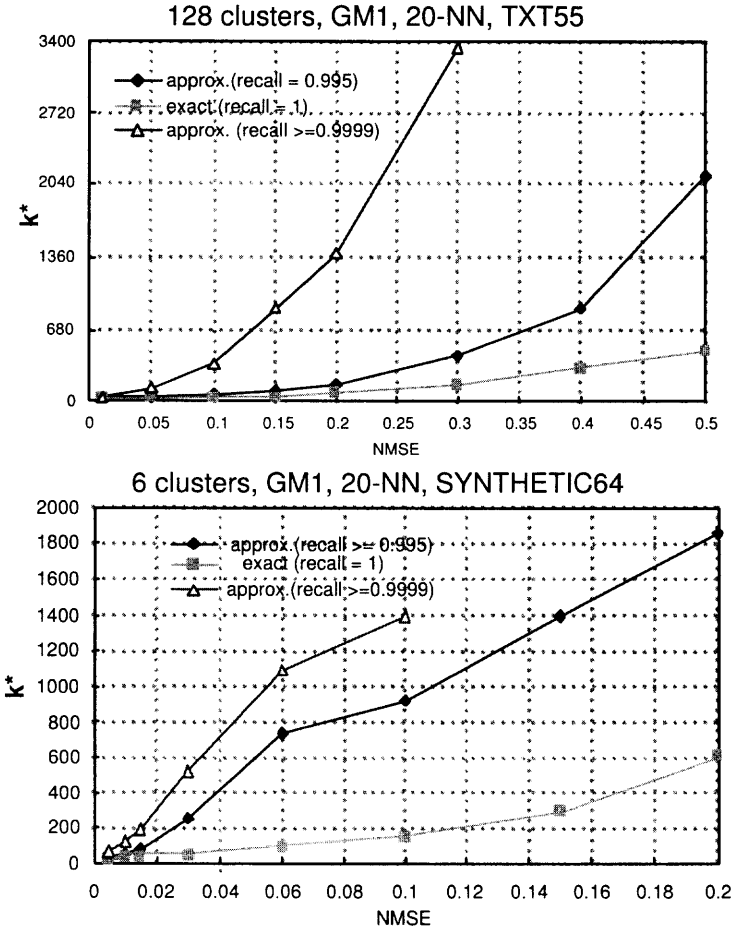


Figure 4.5 Value of k^* for the exact and the approximate algorithm.

on the other hand, the approximate distance does not has this property.

However, since an extra dimension is stored for each point, the CPU time for calculation increases. It can be seen in Figure 4.7, as the NMSE changes from 0.01 to 0.3, although k^* is much smaller, the total CPU cost is higher when using $d+1$ dimensions than when using $d+1$ dimensions. This is only true when sequential scan is used for within-cluster search, when multi-dimensional indexes are used for within-cluster search, the curves of query costs are different. The results with SR-tree can be found in Chapter 5.

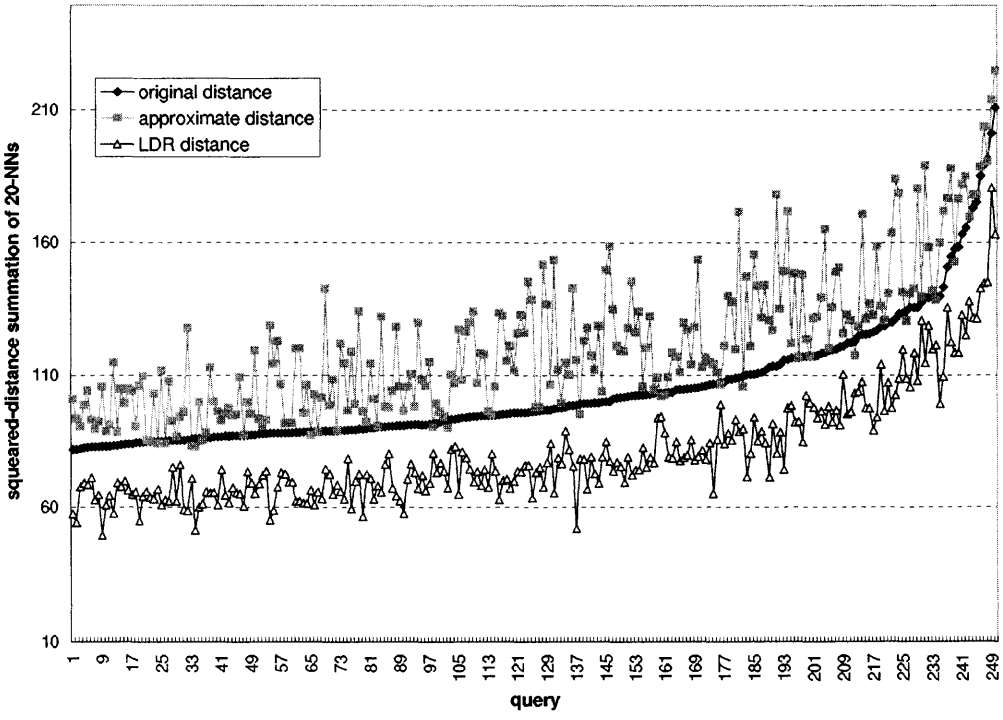
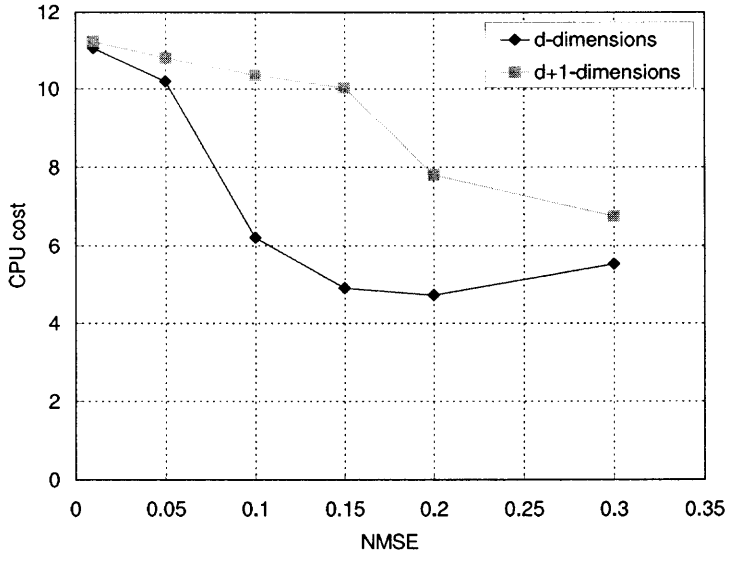


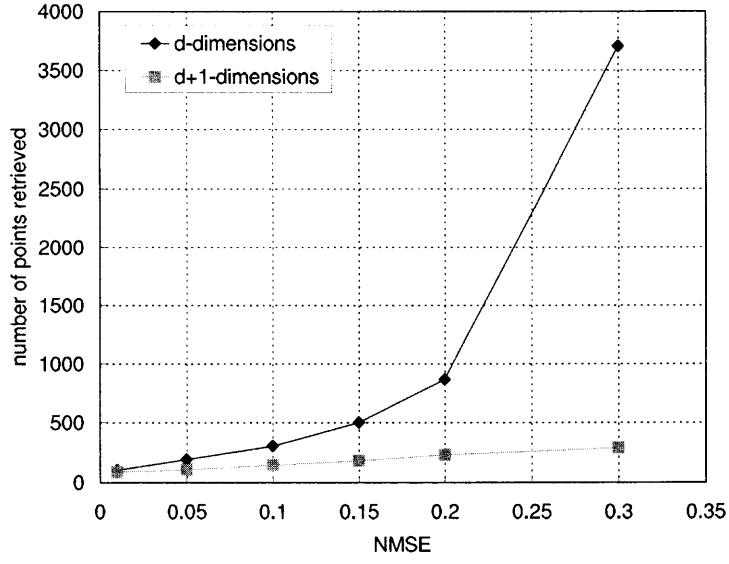
Figure 4.6 The distances between 250 queries and their 20 nearest neighbors, NMSE = 0.2, TXT55. Note that each distance is the sum of squared-distances of the 20 points to the query point.

4.5 Conclusion and Discussion

In this chapter, an algorithm for exact k-NN search on CSVD generated datasets is developed. The exact method ensures a 100% recall, while the approximate method described for the same purpose in [19] can not guarantee an acceptable minimum recall. Furthermore, experimental results show that the exact method has a lower CPU cost than the approximate method, unless the NMSE is very small.



(a)



(b)

Figure 4.7 (a) CPU cost and (b) number of points retrieval ($k^\star$) for d dimensions and $d+1$ dimensions. GABOR60, 5 clusters.

CHAPTER 5

OPTIMAL SUBSPACE DIMENSIONALITY

5.1 Introduction

The k-NN queries have been used in a wide variety of applications, such as Content-Based Retrieval (CBR) from multimedia database and data mining, to find the k most similar objects to the query object. The common ground of these varied applications is that the objects can be described as multi-dimensional points with fixed number of dimensions and the "similarity" of two objects is determined by a *distance metric*, such as the Euclidean distance, between the two corresponding points.

In order to facilitate fast query processing on large multi-dimensional datasets, multi-dimensional indexing structures are used instead of sequential scan. However, as dimensionality increases, multi-dimensional indexing structures degrade rapidly because of the *dimensionality curse* [18]. The problem can be solved by reducing dimensions without losing much information before building an index.

There are many dimensionality reduction methods based on different applications and techniques. One popular method is to utilize linear transformation like Singular Value Decomposition (SVD) or Karhunen-Loeve Transform (KLT) to do Principal Components Analysis (PCA) [41] and rotate and project data points into a lower dimensional space. According to the ways of utilizing SVD or PCA, they can be divided into two categories: *Global* methods and *Local* methods.

Global methods perform SVD or PCA on an entire dataset. Given a dataset X with size M and dimensionality N, one can use SVD to get eigenvalues $\lambda_1 \geq \lambda_2, ..., \geq \lambda_N$ (without loss of generality, they are assumed in a decreasing order) and the corresponding eigenvectors matrix $V = (\vec{v}_1, \vec{v}_2, ..., \vec{v}_N)$ of the covariance matrix of X [19]. Then transform the whole dataset into $Y = XV$. The eigenvectors are also

called the principal components of X. Since they are ordered in a way that the first n dimensions of Y keep most of the variation or energy, the last $N - n$ dimensions can be reduced without losing much information.

Global methods rely on global information derived from the dataset, therefore it is more effective when the dataset is *globally correlated.* In other words, it works well for a dataset whose distribution is well captured by the centroid and the covariance matrix. When the dataset is not globally correlated , i.e., the data points distribution is "heterogeneous" in the dataset – this is often the case for real-world datasets, performing dimensionality reduction using SVD on the entire dataset may cause a significant loss of information, as illustrated by Figure 3.2(a). In this case, there are subsets of the dataset which exhibit local correlation. Local method partitions the large dataset into clusters, and do dimensionality reduction using SVD respectively for each cluster (Figure 3.2(b)).

Local dimensionality reduction methods are usually associated with clustering. CSVD (*Clustering and Singular Value Decomposition*) [72, 19] partitions a dataset into clusters using LBG [55] first and then perform dimensionality reduction in a global manner for all clusters. LDR (*Local Dimensionality Reduction*) [22] starts from spatial clusters and rebuilds clusters by assigning each point to a cluster requiring minimum dimensionality to hold it with an error *ReconDist* below *MaxReconDist.* Another method called MMDR [40] tries to cluster a high-dimensional dataset using the low-dimensional subspace using elliptical k-means clustering based on *Mahalanobis* distance instead of regular k-means clustering which is based on Euclidean distance. The clusters generated by LDR and MMDR should be "SVD friendly" [72] because clusterings are obtained based on error thresholds after projecting points into subspaces.

Range queries and k-NN queries are the two most popular query types for similarity search. k-NN queries retrieve k closest data objects to the query object and range queries return all the data objects within a distance ϵ to the query object. k-NN

search methods can be divided into two categories: *exact* methods ([66, 37, 48, 68]) and *approximate* methods ([31, 7, 19]). Exact k-NN search returns the exact k closest points which appear the same as the results of linear searching on the original dataset, while approximate k-NN search returns approximate results and guarantees a certain accuracy.

No matter which k-NN search method is considered, it is a critical issue to decide the number of dimensions to be retained. Few of the multi-dimensional indexing structures perform well when dimensionality exceed 30. On the other hand, too much dimensionality reduction results in too much distance information loss. The challenge here is to find a tradeoff between dimensionality reduction and information loss. Nothing that in this chapter, only exact k-NN method is considered.

It has been observed that when the index is created on dimensionality reduced data, the cost of a query ($Cost_q$) is composed of two parts: the cost of querying the index ($Cost_i$) and the cost of post-processing ($Cost_p$), i.e., the cost of removing *false alarms*, which are non-qualified points. Therefore,

$$Cost_q = Cost_i + Cost_p \tag{5.1}$$

$Cost_i$ decreases as more dimensions are moved, but at the same time, $Cost_p$ increases because of more distance information lost. There must be a point or an interval of dimensions in which the minimum $Cost_q$ is reached, e.g., point A and interval [B, C] in Figure 5.1.

The query cost has been studied through modeling for specific indexing structures. However, the previous work has the following limitations:

- Only considered the cost for index querying.
- Only can be used for global dimensionality reduction methods.

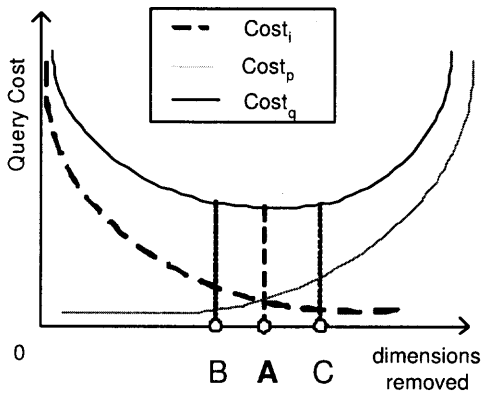


Figure 5.1 Optimal subspace dimensionality with respect to minimal query cost.

- Failed to consider all side effects when dimensionality gets higher.

Local dimensionality reduction methods generate multiple clusters so that each of them may have different subspace dimensionality respectively, thus to find optimal subspace dimensionality with respect to minimum query cost becomes much more difficult.

In this chapter, a hybrid method is presented which takes advantage of the clustering algorithm of existing local dimensionality reduction methods and removes dimensions from each cluster according to the ratio of information loss. Through this method, the relationships among query cost, ratio of total information loss, and subspace dimensionality of each cluster is discovered so that not only optimal subspace dimensionality can be determined, but also users can have more freedom to make tradeoff between query cost and index size. The new method is based on experiments, i.e., perform off-line experiments before real applications.

The rest of the paper is organized as follows. Section 5.2 introduces related work. In Section 5.3, the new method is presented in detail. Experiments are given in Section 5.4 and conclusion is given in Section 5.5.

5.2 Related Work

The cost of a similarity query consists of CPU cost and I/O cost. For memory-resident indexing methods or linear scan, usually only CPU cost is considered, while for disk-resident indexing structures, I/O cost, measured as the number of page accesses, is typically considered. The estimations of k-NN query cost and range query can be transformed to each other by estimating the *selectivity* of a range query and the approximate query radius of a k-NN query.

The problem of modeling query cost for multi-dimensional index structure has been studied for many years [28, 12, 16, 47]. Faloutsos et al. [28] present a model to analysis the range query cost for R-tree [36]. Then in [47] the cost model for k-NN search is given for two different distance metrics. They both consider effect of correlation among dimensions by utilizing *fractal dimensions*, but the models are limited to low-dimensional data space. A cost model for high-dimensional indexing proposed in [16] takes into account the *boundary effects* of datasets in high-dimensional space. But it assumes the index space is overlap-free, which is impossible in high-dimensional space for most of the popular indexing structures. In fact, as the dimensionality increases, the overlap among the nodes of a spatial tree structure is getting so heavier that can not be ignored. Therefore, the cost models can not estimate the real cost of querying for high-dimensional datasets. In additional, the cost models above only focus on the indexing cost, they fail to consider the cost due to dimensionality reduction. Detailed information about fractal dimensions and the above query models is provided in Appendix C.

In this thesis, the cost of k-NN queries is analyzed through experiments. The author of this thesis chooses not to analyze index query cost by modeling because:

1. when dimensionality gets higher, there are so many side effects that mathematical formulation can hardly consider all of them and the precision can not be guaranteed. The equations in [16] are already very complicated although it

only consider some of the effects.

2. Most of the cost models assume the input of data points are *bulk loading* or *packing*, which means divide the dataset into rather small regions which fit into a single data page so as to reduce overlap. One of the commonly used techniques is the *space filling curve* like Hilbert curve [28] and Z-ordering. Another technique is to divide the data space into partitions which correspond to data pages, either in a top-down manner [49] or bottom-up manner [17]. However, although they have very good performance for low-dimensional data, the packing algorithms are not efficient for high-dimensional datasets. Moreover, as most modern applications require dynamic insertion and deletion, indexes should be created dynamically, rather than statically. Some results in Appendix C show that the cost models do not work for datasets which are not preprocessed.

3. It is shown that the index querying cost is strongly related to the ratio of information loss in this chapter, therefore the approximate interval of subspace dimensionality with respect to minimum query cost is expectable and therefore estimating query cost by experiments is applicable.

The CSVD is also a hybrid method that combines clustering and dimensionality reduction, but it uses LBG for clustering which can not be able to identify locally correlated clusters and furthermore, it fails to provide any discussion on optimal subspace dimensionality.

High-dimensional clustering method like CLIQUE [4] discovers clusters embedded in subspaces, but it can not be used as the first step of the new method because it can only find correlation along the dimensions in the original space.

5.3 The Hybrid Method for Clustered Datasets

In this chapter, the data being processed are not original datasets but the resulting datasets after clustering or even after dimensionality reduction, as shown in Figure 5.2.

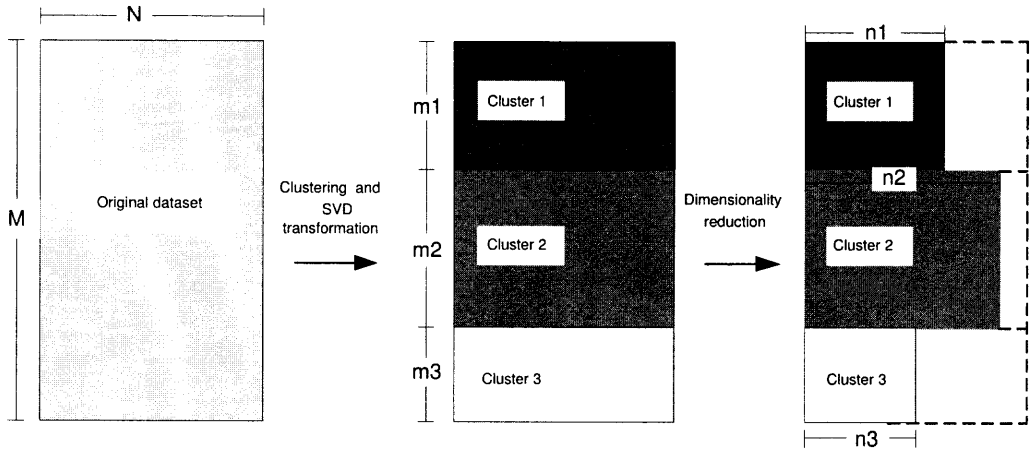


Figure 5.2 Clustered dataset and Dimensionality reduced clustered dataset.

Definition 5.3.1 *A* **Clustered Dataset** X_C *is generated by a clustering method and is specified as* $X_C = \{X_1, X_2, ..., X_H\}$, *where* H *is the number of clusters. The size of cluster* X_h $(h \in \{1, \ldots H\})$ *is* m_h *and* $\sum_{h=1}^{H} m_h = M$.

Definition 5.3.2 *A* **Dimensionality Reduced Clustered Dataset** X_{DRC} *differs from* X_C *in that the dimensionality of* X_h *is* n_h $(n_h \leq N)$. *The average dimensionality is defined as* $\bar{n} = \sum_{h=1}^{H} n_h \times m_h/M$.

5.3.1 Information Loss and Subspace Dimensionality

The LDR generate clusters based on local correlation inside a dataset and therefore is efficient for locally correlated dataset. However, according to the LDR method, the subspace dimensionality of each cluster is determined by many factors, like *MaxReconDist*, *FracOutliers*, *local_threshold* and the threshold of cluster size – *MinSize*. These parameters must be carefully selected to obtain "good" results and usually it

takes many iterations to figure them out for a dataset. While, whether or not a result is "good" relies on user's demands. Dimensionality reduction results in information loss and users care more about the ratio of total information loss rather than individual information loss (*ReconDist* in LDR) because the ratio of total information loss is a very important link between the subspace dimensionality and efficiency of query processing, e.g., how does it affect the CPU cost of k-NN queries when 20 out of 64 dimensions are reduced? It depends on how much information loss.

Normalized Mean Squared Error – NMSE

It is known that the total distance information loss is directly related to the subspace dimensionality when SVD is performed on the whole dataset: the more dimensions retained, the less distance information lost. But for clustered dataset, how does the dimensionality reduction of individual cluster affects the total information loss and how to control the individual subspace dimensionality according to the error user can tolerate? If these questions are answered, then an optimal choice of subspace dimensionalities can be decided to ensure efficiency.

To solve this problem, a formal description of the total distance information loss – Normalized Mean Squared Error (NMSE)– is needed. It calculates the ratio of the total distance information loss after dimensionality reduction to the total distance information before dimensionality reduction.

$$NMSE = \frac{\sum_{h=1}^{H}\sum_{i=1}^{m_h}\sum_{j=n_h+1}^{N}(y_{i,j}^{(h)})^2}{\sum_{h=1}^{H}\sum_{i=1}^{m_h}\sum_{j=1}^{N}(y_{i,j}^{(h)})^2} \tag{5.2}$$

Equation 2.6 and Equation 2.6 in Chapter 2 gives the definition of NMSE on a dataset with size M and dimensionality N. Since SVD is a linear transformation that does not change the Euclidean Distance between points, the distance information loss of transformed dataset is equal to that of the original dataset. Applied to clustered datasets Equation 5.2 is obtained, where H is the number of clusters and m_h and n_h

refer to the size and dimensions retained for X_h.

There is a simpler way to calculate NMSE. As shown in Equation 2.7 (Chapter 2), the NMSE of a single cluster is also equal to the ratio of summation of retained eigenvalues to the summation of all eigenvalues, where λ_j is the j-th largest eigenvalue.

Based on Equation 2.7 and Equation 5.2, the equation of the NMSE for clustered datasets (Equation 3.4) is obtained, where $\lambda_j^{(h)}$ is the j-th largest eigenvalue of X_h. The proofs of Equation 2.7 and Equation 3.4 can be found in [73].

Dimensionality Reduction According to the NMSE

Equation 3.4 can be also summarized as a function between NMSE and subspace dimensionality of each cluster: $NMSE = f(n_1, n_2, ..., n_H)$. For such a function, given a target NMSE (*tNMSE*), there are many solutions for choosing $n_1 \sim n_H$. CSVD has a good approach to reduce dimensionality of clustered datasets in a global manner by sorting eigenvalues of all clusters and removing least significant ones until *tNMSE* is reached. The algorithm uses a min-priority queue Q which has three fields for each entry: the eigenvalue (key), cluster ID it belongs to and the dimension ID associated with it. The algorithm is shown in Table 5.1, named Algorithm 1. This method has been named *GM1* in Chapter 3, while the details of the algorithm has not been given there.

It follows Equation (3.4) to calculate *currentNMSE*, therefore the resulting set of dimensions $n_1 \sim n_H$ satisfies $tNMSE$.

Step 1, 2 3 and 4 finish the initialization, where *sumBase* and *sumRemoved* are used to store and compute the denominator and numerator of Equation (3.4) respectively. Step 5, 6 and 7 remove the least significant eigenvalue from the min-priority queue and updates *currentNMSE* iteratively until the queue becomes empty or the *currentNMSE* is large enough. The subspace dimensionality of each cluster is updated and returned in step 8 and 9.

After performing Algorithm 1, $NMSE = f(n_1, n_2, ..., n_H)$ becomes a one-to-one

Table 5.1 Algorithm 1

Input: (1)The size m_h and eigenvalues $\lambda_1^{(h)} \geq \lambda_2^{(h)} \geq ... \geq \lambda_N^{(h)}$ for X_h , $h = 1, ..., H$. (2)$tNMSE$.

Output: n_1 ... n_H.

1. Push eigenvalues of all clusters into queue Q.
2. $sumBase \leftarrow \sum_{h=1}^{H} \sum_{j=1}^{N} m_h \times \lambda_j^{(h)}$.
3. $sumRemoved \leftarrow 0$.
4. For $h = 1$ to H, $currentDim_h \leftarrow N$.
5. If Q is empty goto step 7, otherwise pop $\lambda_i^{(h)}$ from Q.
6. $sumRemoved \leftarrow sumRemoved + m_h \times \lambda_i^{(h)}$.
7. $currentNMSE \leftarrow sumRemoved/sumBase$.
8. If $currentNMSE \leq tNMSE$, $currentDim_h \leftarrow currentDim_h - 1$, goto step 5.
9. For $h = 1$ to H, $n_h \leftarrow currentDim_h$.

monotonous function. Then once the relation between query cost and NMSE is found, the optimal subspace dimensionality can be obtained right away. Since there are at most $H \times N$ items in the queue, and the number of clusters H is a constant usually smaller than 100, the cost of Algorithm 1 is just $O(N)$.

5.3.2 The Hybrid Method

The hybrid method is shown in Table 5.2. Step 1 uses a local dimensionality reduction

Table 5.2 Algorithm 2

1. Perform the clustering algorithm of a local dimensionality reduction method (like LDR) to obtain clusters and principal components. Ignore the subspace dimensionality it generated.
2. For $tNMSE = start$ to end step β
   (a) Perform Algorithm 1 for dimensionality reduction according to $tNMSE$ to obtain the function of $tNMSE = f(n_1, n_2, ..., n_H)$.
   (b) Create index for each cluster in subspace using any existing multi-dimensional indexing structure.
   (c) Perform k-NN search with sample queries and record I/O cost.
3. Plot the I/O cost versus NMSE and get the value of NMSE corresponding to the minimum cost, then the corresponding set of subspace dimensionalities is the answer.

method to generate local correlated clusters. If the $tNMSE$ is specified by users, step 2 only has one iteration. Otherwise, the value *start*, *end* and β have to be decided. According to experiences, *start* could be 0.005 and *end* is not necessary to be larger

than 0.3. If you want a precise answer the step β should be small enough, on the other hand, if you want just an approximate interval of NMSE with respect to minimum query cost, β can be larger, which can reduce the cost of the algorithm. In the experiments of this chapter, β is set such that $\frac{end-start}{\beta} = 8$.

Actually you don't need to have more than five iterations for a dataset since the experiments in Section 5.4 show that minimal query cost always fall in a small range of NMSE from 0.05 to 0.35, no matter what kind of dataset is involved.

The subspace dimensionality LDR generated is ignored because it is useless when iterations are involved. Suppose only *MaxReconDist* is varied. Then each time *MaxReconDist* is changed, the whole process including clustering has to be redone completely to obtain another set of subspace dimensionality. While using the proposed method, the dataset only needs to be clustered once and then vary NMSE to generate subspace dimensionality using Algorithm 1 without even touching any data point.

5.3.3 Optimal Subspace Dimensionality

Searching index structures built on dimensionality reduced datasets results in *false alarms*, which can be removed by accessing the original dataset to obtain the original distances between the query and candidate nearest neighbors. The distance between projected (dimensionality reduced) points lower bounds the distance between the original points, therefore there is no *false dismissal* [27].

According to Equation 5.1, when indexes are created on an original dataset, $Cost_p$ is zero, but $Cost_i$ is high and therefore the query cost $Cost_q$ is quite high because of the dimensionality curse. $Cost_i$ drops as more dimensions are reduced, but $Cost_p$ increases because of more *false alarms* as the NMSE increases. When too many dimensions are removed, $Cost_i$ becomes low; however, too much distance information is lost and $Cost_p$ becomes very high, therefore $Cost_q$ is very high. Consequently,

$Cost_q$ is like a convex function of the NMSE. There must be an optimal subspace dimensionality at which the query cost is the lowest. This optimal subspace dimensionality could be affected by indexing methods or dimensionality reduction methods, but for a given index and a given dimensionality reduction method, it should be only related to the NMSE.

Besides query cost, index size is another concern of users. The more dimensions being removed, the smaller the index is. User may want a subspace dimensionality such that the index size does not exceed a given value, although the query cost might not be the lowest. Therefore, the more general objective function should be described as:

$$Cost_{total} = \alpha \cdot Cost_q + (1 - \alpha) \cdot index\_size \tag{5.3}$$

where $\alpha \in (0, 1]$ is an weight factor and $index\_size = \sum_{h=1}^{H} m_h \cdot n_h$.

If only query cost is considered, just set α to 1. If index size is also considered, tradeoff can be made between index size and query cost by set α to a certain value between 0 and 1.

Of course no matter through modeling or experimental analysis, it is not easy to find the exact optimal value of subspace dimensionality with respect to the minimum query cost. Instead, it is practical and still valuable to find a small interval and make sure the optimal value is in that interval.

5.4 Experiments

In this section, several experiments are carried out for the hybrid method on four datasets (three real-world datasets and one synthetic dataset), as shown in Table 5.3. The three real-world datasets are all image datasets with different feature vectors. The synthetic dataset is created to have local correlation along arbitrary directions using the algorithm in [22].

Table 5.3 Datasets for the Experiments

Name ($M \times N$)	Description/Source
TXT55 ($79,814 \times 55$)	Gabor, spatial, and wavelet features, from 400 photos. Obtained from [19].
GABOR60 ($56,644 \times 60$)	Gabor features extracted from MMS images from different parts of the country. Obtained from Dr. V. Castelli.
SYNT64 ($99,984 \times 64$)	Synthetic dataset generated to have 5 local correlated clusters in subspaces of different orientations by using the algorithm from [22].
COLH64 ($68,014 \times 64$)	8×8 color histograms extracted from 68,014 color images obtained from http://kdd.ics.uci.edu/databases/CorelFeatures.

LDR is utilized to generate clusters. For each dataset, two clustered datasets are generated by varying LDR parameters. The detailed information of those clustered datasets are given in Table 5.4. In order to find optimal subspace dimensionality, the NMSE is varied gradually from 0 to 0.5 for each clustered dataset.

Table 5.4 Clustered Datasets (CD) Generated by LDR with Different Parameters

Datasets	CD	LDR parameters				H
		local_threshold	*FracOutliers*	*MinSize*	*MaxReconDist*	
SYNT64	1	1.0	0.15	800	0.7	9
	2	1.0	0.15	800	0.8	8
GABOR60	1	3.0	0.15	300	1.0	5
	2	3.0	0.15	300	2.0	4
TXT55	1	10.0	0.15	800	4.0	7
	2	10.0	0.15	800	6.0	6
COLH64	1	2.0	0.15	800	0.25	4
	2	2.0	0.15	800	0.35	3

SR-trees [42] is used as the indexing method because it was shown to be more efficient than R-trees in high dimensional space. The split factor for SR-trees is 0.4 and the reinsert factor is 0.3. Before run the algorithm on SR-trees, some experiments has been done using sequential scan as within-cluster searching methods.

For sequential scan, the exact k-NN algorithm proposed in Chapter 4 is applied, while when using SR-trees, the exact k-NN algorithm of LDR is applied (See Appendix A), which has been proved to be optimal [22]. It uses a priority queue to navigate the tree structures of all clusters and finds the exact answers by ranking. Therefore, it accesses as few points as possible.

The query costs appeared in all figures are averaged over one thousand 20-NN

queries which are randomly selected from each dataset.

The experiments were implemented using C++ on a Dell Workstation (Intel Pentium 4 CPU, 2.0 GHz, and 512 MB RAM) with Windows 2000 Professional.

5.4.1 NMSE and the Average Subspace Dimensionality

The datasets are clustered datasets with more than one cluster each. To make life easier, the average subspace dimensionality defined in Section 5.3.1 is plotted instead of dimensionality of each cluster. Besides the reason specified at Section 5.3.2, the subspace dimensionality obtained by LDR is ignored for the following two points:

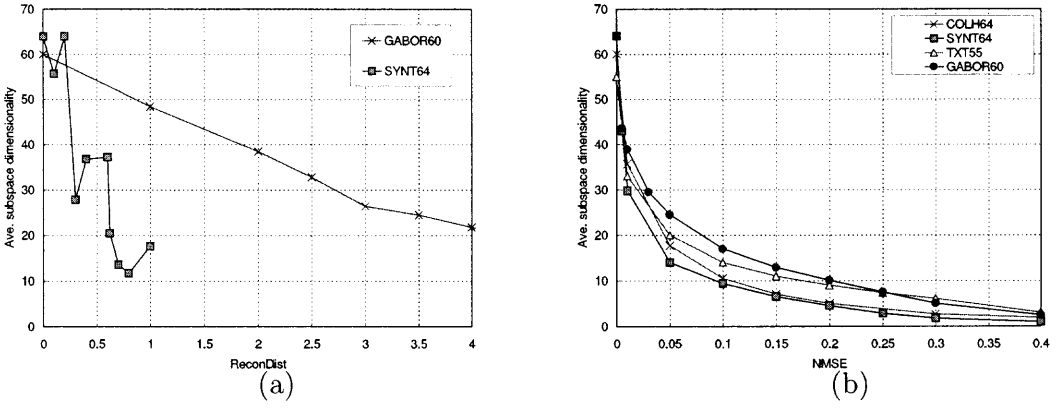


Figure 5.3 Average subspace dimensionality vs. (a)LDR threshold *ReconDist* for the clustered datasets of GABOR60 and SYNT64. (b)NMSE (by using Algorithm 1) for the clustered datasets of all four datasets.

- The average subspace dimensionality can not always be described as a monotonous function of *MaxReconDist*, see the series of SYNT64 in Figure 5.3 (a), even if all other parameters are fixed. Also it can be seen from Table 5.4 that the number of clusters changes as well. The curves in Figure 5.3 (b) show that, when using Algorithm 1, the average subspace dimensionality decreases monotonously and smoothly as NMSE increases, although each dataset may have different paces.

- For some datasets the curve of average subspace dimensionality with respect

to *MaxReconDist* is monotonously decreasing like the series of GABOR60 in Figure 5.3 (a), but the magnitude of *MaxReconDist* is unexpectable. It can be seen from Figure 5.3 (a), for SYNT64 the average subspace dimensionality is 18 at $MaxReconDist = 1$, while for GABOR60, LDR does not have such level of dimensionality reduction until $ReconDist \geq 4$. Consequently, the choice of LDR parameters is very much data-dependent. Unlike *MaxReconDist*, NMSE is always in [0, 1].

5.4.2 Query Costs and Subspace Dimensionality

A. Without High-dimensional Indexing

The following are the experiments without building any indexes, just linear scanning among the clusters. Figure 5.4 plots the CPU costs versus NMSE for SYNT64 and GABOR60. The results show that the curves are just like what are expected and the optimal dimensionality is always obtained at around 5% to 20% information loss, i.e., when NMSE $\in [0.05, 0.2]$, no matter what kind of dataset is involved.

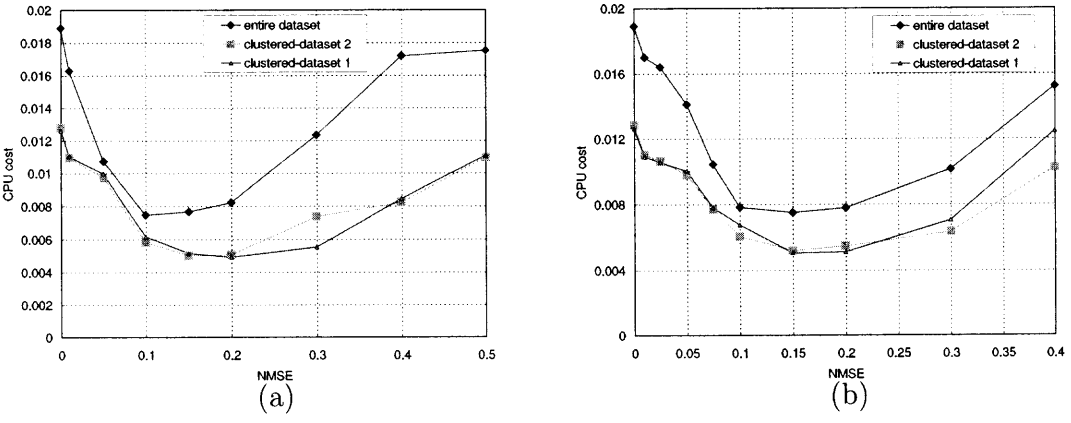


Figure 5.4 CPU costs of 20-NN queries vs. NMSE for (a) SYNT64 (b) GABOR60.

B. With SR-trees

Algorithm 2 is implemented on the clustered datasets in Table 5.4 for the four datasets respectively. The results are listed in Figure 5.5, 5.6, 5.7 and 5.8, where (a) of them illustrate query costs versus NMSEs, and (b) of them illustrate query costs versus subspace dimensionalities. The query costs are the summation of index query costs and post-processing cost which are both measured as the number of page accesses. Like the LDR data structure, $d + 1$ dimensions is kept when the dimensionality is reduced to d, where the reconstruction distance *ReconDist* is stored in the extra dimension. The following conclusions are obtained from the experimental results:

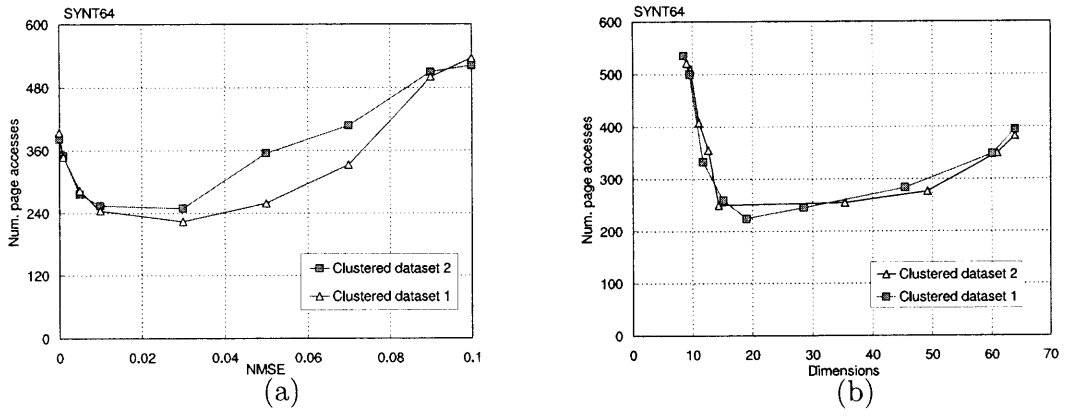


Figure 5.5 I/O costs of 20-NN queries vs. (a)NMSE and (b)subspace dimensions for SYNT64 with SR-trees.

- The curve between query cost and average subspace dimensionality has a "U" shape as expected, and optimal subspace dimensionality does exist.
- Different clustered datasets from the same dataset have almost same optimal intervals of subspace dimensionality.
- Different datasets have different optimal subspace dimensionalities since the correlation degrees are different. But they have almost same NMSE values with

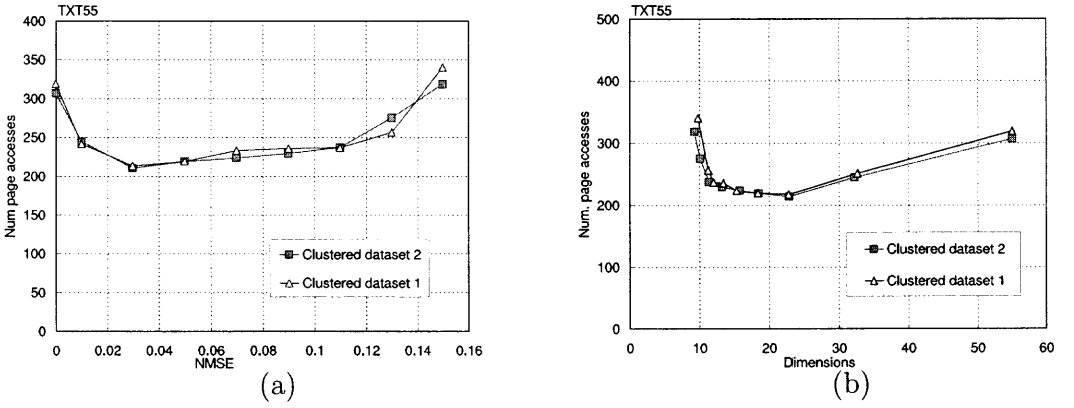


Figure 5.6 I/O costs of 20-NN queries vs. (a)NMSE and (b)subspace dimensions for TXT55 with SR-trees.

respect to the minimum query costs. From the four figures it can be seen that the optimal subspace dimensionalities attain at around 20 for SYNt64 (Figure 5.5 (b)), 23 for TXT55 (Figure 5.6 (b)), 18 for COLH64 (Figure 5.5 (b)) and 32 for GABOR60 (Figure 5.5 (b)), but the minimum costs all attains at around $NMSE = 0.03$(Figure 5.5 (a), 5.6 (a), 5.7 (a) and 5.8 (a)). It means minimum query cost is strongly related to NMSE.

With SR-trees, the minimum k-NN query cost is achieved at $NMSE \approx 0.03$ for LDR generated clustered datasets, no matter on synthetic dataset or real-world image datasets. In other cases, i.e., using other indexing methods, or dimensionality reduction methods, or other kind of datasets, the value of NMSE might be different.

5.5 Conclusion

In this paper, a hybrid method is presented to discover the relationship among k-NN query cost, ratio of total information loss, and subspace dimensionality for clustered datasets, which are generated by a local dimensionality reduction method. The author found that the optimal subspace dimensionality does exist and can be identified through the proposed method. In addition, it can be seen that the minimum query

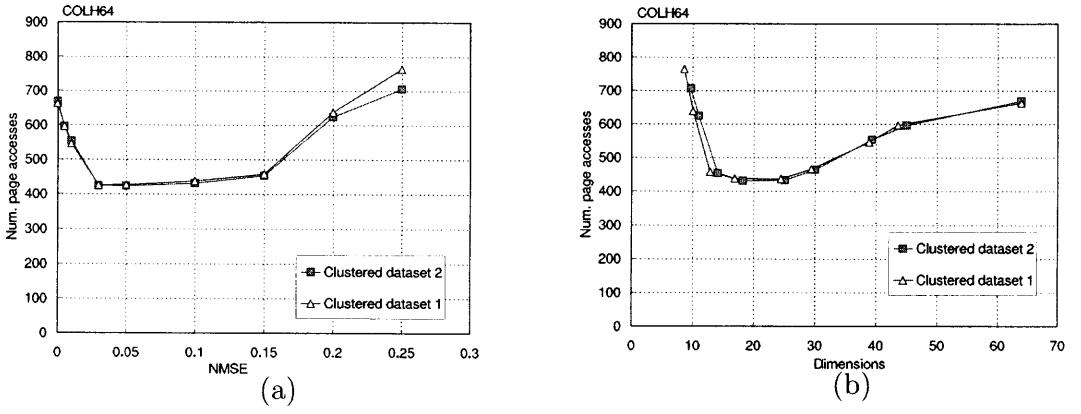


Figure 5.7 I/O costs of 20-NN queries vs. (a)NMSE and (b)subspace dimensions for COLH64 with SR-trees.

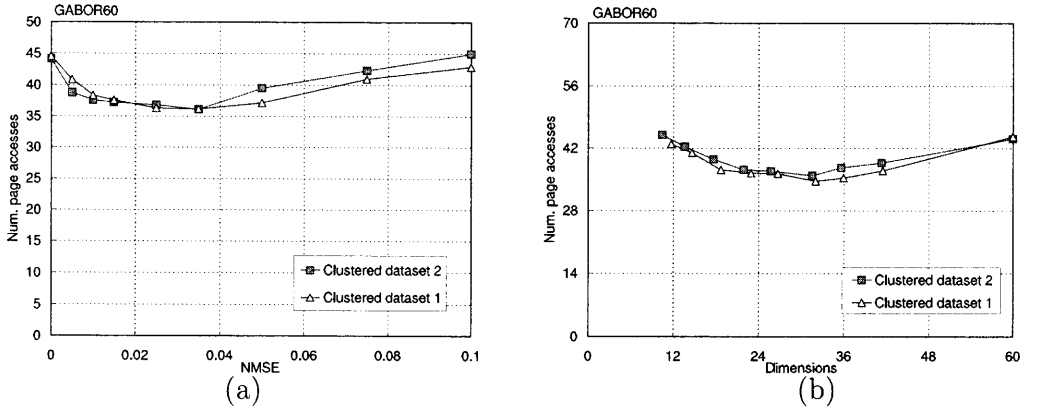


Figure 5.8 I/O costs of 20-NN queries vs. (a)NMSE and (b)subspace dimensions for GABOR60 with SR-trees.

cost is strongly related to the NMSE for a given index method and dimensionality reduction method, and therefore using the NMSE for finding optimal subspace dimensionality is a good choice. The experiments show that the new method works well for both real-world datasets and synthetic datasets.

CHAPTER 6

CONCLUSION

Similarity search in high-dimensional datasets is very important for many modern database applications. In order to deal with the dimensionality curse as the dimensionality increases, many techniques such as multi-dimensional indexing and dimensionality reduction, have been introduced to improve the efficiency of similarity search. The contributions of this thesis can be summarized as follows:

- **Improvements on CSVD:** Dimensionality reduction, using the Singular Value Decomposition method to retain a subset of features which are supposed to be more *important*, is considered to be an effective way to improve the efficiency of index structures. The SVD or PCA have been widely used for identifying the principal components of the original datasets to which dimensionality reduction is applied. This method is very effective when the dataset consists of *homogeneously* distributed vectors. For *heterogeneously* distributed vectors, or local correlated datasets, a more efficient representation, with the same degree of normalized mean squared error, can be generated by CSVD – dividing the dataset into clusters, reducing dimensionality individually using SVD. In this thesis, the three methods for selecting dimensions to be retained for CSVD (LM, GM1 and GM2) have been presented and compared with each other. Experimental results with four datasets show that GM1 outperforms LM and GM2.

- **Performance comparison of local dimensionality reduction methods:** In this thesis, local methods CSVD and LDR have been analyzed and compared from the viewpoints of compression ratio, CPU cost, and retrieval efficiency. MMDR was also compared to CSVD from the viewpoints of compression ratio

and CPU cost on synthetic as well as real-world datasets. Experiments are held by using sequential scan for within-cluster search and the results show that CSVD outperforms LDR and MMDR.

- **An exact k-NN search algorithm:**

  An algorithm to find the exact k nearest neighbors has been proposed for local dimensionality reduction methods. The original k-NN algorithm for CSVD is an approximate method since it violates the lower-bounding property. The proposed k-NN algorithm is based on a multi-step k-NN search algorithm which is designed for global dimensionality reduction method. Experiments with two datasets show that it requires less CPU time than the approximate algorithm at a comparable level of accuracy.

- **Optimal subspace dimensionality for clustered datasets:**

  Since dimensionality reduction cause distance information loss, the number of dimensions to be retained becomes a critical issue. The total cost of a similarity query is the sum of index query cost and postprocessing cost, which is used to remove false alarms. As the number of dimensions being removed increases, index query cost decreases obviously, however, postprocessing cost increases due to the increasing number of false alarms. There must be an optimal subspace dimensionality at which the query cost is minimized. Local dimensionality reduction methods generate multiple clusters and each of them has different subspace dimensionality respectively. This makes the identification of optimal subspace dimensionalities more difficult. In this thesis a hybrid method has been presented to determine optimal subspace dimensionality of each cluster. The experiments on four datasets show that the proposed method works well for both real-world datasets and synthetic datasets.

APPENDIX A

K-NN ALGORITHM OF LDR

The algorithm for k-NN queries is shown in the Table A.1. It was proposed for LDR method in [22]. It uses a min-priority queue (*queue*) to navigate the nodes and objects in the multi-dimensional indexes of the clustered dataset in increasing order of their distances from query Q. Each entry in the queue is either a node or a point and stores: the *id* of the node or point T it corresponds to, the cluster S it belongs to and its distance *dist* from the query anchor Q. The items are sorted on dist i.e., the smallest item appears at the top of the queue. For nodes, the distance is defined by *MINDIST*, while for objects, it is just the point-to-point distance.

Initially, for each cluster S_i, Q is mapped to its subspace using the information stored in the root node R_i and becomes Q_i. Then, for each cluster, the distance $MINDIST(Q_i, R_i)$ of Q_i from the root node R_i is computed and pushed into the queue along with the distance and the id of the cluster S_i. The k closest neighbors of Q among the outliers are calculated with sequential scan and are put into the set *temp*.

Step 5 through Step 19 are the navigation of the indexes (one SR-tree for each cluster for example). An item is popped out from the top of the queue at each outer loop. If the popped item is a point, compute the distance of the original E-dimensional point (by accessing the full tuple on disk) from Q and append it to *temp* (Lines 11-13). If it a node, compute the distance of each of its children to the appropriate projection of Q - $Q_{top.S}$ (where *top*.S denotes the cluster which *top* belongs to) and push them into the queue (Lines 14-19). A point O is moved from *temp* to *result* only when it is among the k nearest neighbors of Q for sure. The condition $O.dist \leq top.dist$ in Line 7 ensures that there exists no unexplored point O' such that $D(O', Q) < D(O, Q)$.

Table A.1 k-NN Algorithm for LDR

```
1   for(i = 1; i ≤ K; i++)
2       Q_i ← map Q to S_i
3       queue.push(S_i, R_i, MINDIST(Q_i, R_i));
4   Fill temp with the k closest neighbors of Q among outliers
5   while (queue is not empty)
6       top = queue.Pop();
7       for each point O in temp such that O.dist ≤ top.dist
8           temp ← temp - O;
9           result ← result ∪ O;
10          if (|result| == k) return result;
11      if top.T is a point
12          top.dist = D(Q, top.T);//original distance
13          temp ← temp ∪ top.T;
14      else if top.T is a leaf node
15          for each point O in top.T
16              queue.push(top.S, O, D(Q_top.S, O));
17      else //if top.T is an inner node
18          for each child C of top.T
19              queue.push(top.S, C, MINDIST(Q_top.S, C));
```

By inserting the points in *temp* (i.e. already explored items) into *result* in increasing order of their distances in the original space (by keeping *temp* sorted), it also ensure there exists no explored point O' such that $D(O', Q) < D(O, Q)$. This shows that the algorithm returns the correct answer i.e., the same of points as querying in the original space.

This algorithm is applied in Chapter 5 of this thesis.

APPENDIX B

CPU COST OF REGULAR K-MEANS AND ELLIPTICAL K-MEANS

The regular k-means method tends to discover clusters with spherical shapes. In applications like image processing and pattern recognition, it is often desirable to find natural clusters. Data points that are locally correlated should be grouped into one cluster. The elliptical k-means algorithm discovers elliptical shaped clusters which is based on an adaptively changing *normalized Mahalanobis* distance metric as shown in Equation 2.1 in Chapter 2.

CPU cost of the regular k-means and the elliptical k-means is compared in this appendix because, it is done with exact k-NN search algorithm described in Chapter 4 but not the approximate k-NN in Chapter 3, and it didn't produce any significant results.

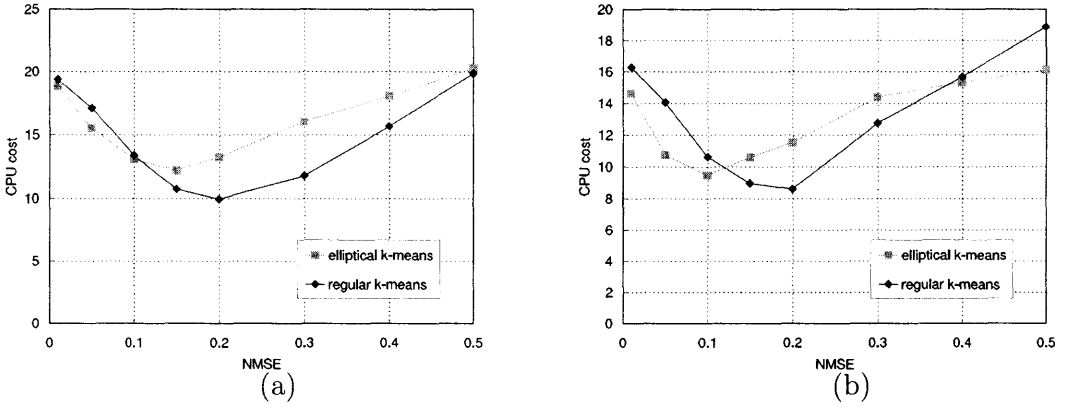


Figure B.1 CPU cost versus NMSE of exact k-NN algorithm for the two k-means algorithm for TXT55. (a): 4 clusters (b): 16 clusters.

Figure B.1 shows the CPU costs of querying clustered datasets produced by regular k-means and elliptical k-means algorithm, with exact k-NN algorithm described in Chapter 4. Actually for the TXT55 dataset, firstly, there is not big difference

between the CPU costs of the two clustering methods, this is the reason that the author uses regular k-means for CSVD in most cases. Secondly, minimum query costs are reached at different points of NMSE for the two methods (0.1 for elliptical k-means and 0.2 for regular k-means). When $NMSE < 0.12$, the elliptical k-means has lower query cost; when $0.12 \leq NMSE < 0.38$, the regular k-means has lower cost; when $NMSE \geq 0.38$, the elliptical k-means has lower query cost again.

APPENDIX C

ANALYTIC K-NN QUERY COST MODEL

The problem of modeling query cost for multi-dimensional index structure has been studied for many years. Faloutsos et al [28] present a model to analyze the range query cost for the R-tree [36]. Then the cost model for k-NN search for two different distance metrics is given in [47]. Both models consider effect of correlation among dimensions by utilizing *fractal dimension*, however, they are limited to low-dimensional data space. A cost model for high-dimensional indexing proposed in [16] takes into account the *boundary effects* of datasets in high-dimensional space. But it assumes the index space is overlap-free, which is impossible in high-dimensional space for most of the popular indexing structures.

The cost model to be verified in this chapter are based on the above two models, therefore they are introduced in details in the following sections. Since they both utilize fractal dimensions, it is necessary to first give a brief introduction to fractals and fractal dimensions.

C.1 Fractal Dimensions

A fractal is a rough or fragmented geometric shape that can be subdivided into parts, each of which is (at least approximately) a reduced-size copy of the whole. Fractals are generally self-similar and independent of scale.

There are many mathematical structures that are fractals; e.g. the Sierpinski triangle, the Koch snowflake, the Mandelbrot set and the Fern (Figure C.1). Fractals also describe many real-world objects, such as clouds, mountains, turbulence, and coastlines, that do not correspond to simple geometric shapes.

A set of points is a fractal if it exhibits self-similarity over all scales. This is illustrated by an example: Figure C.2 shows the first few steps in constructing the

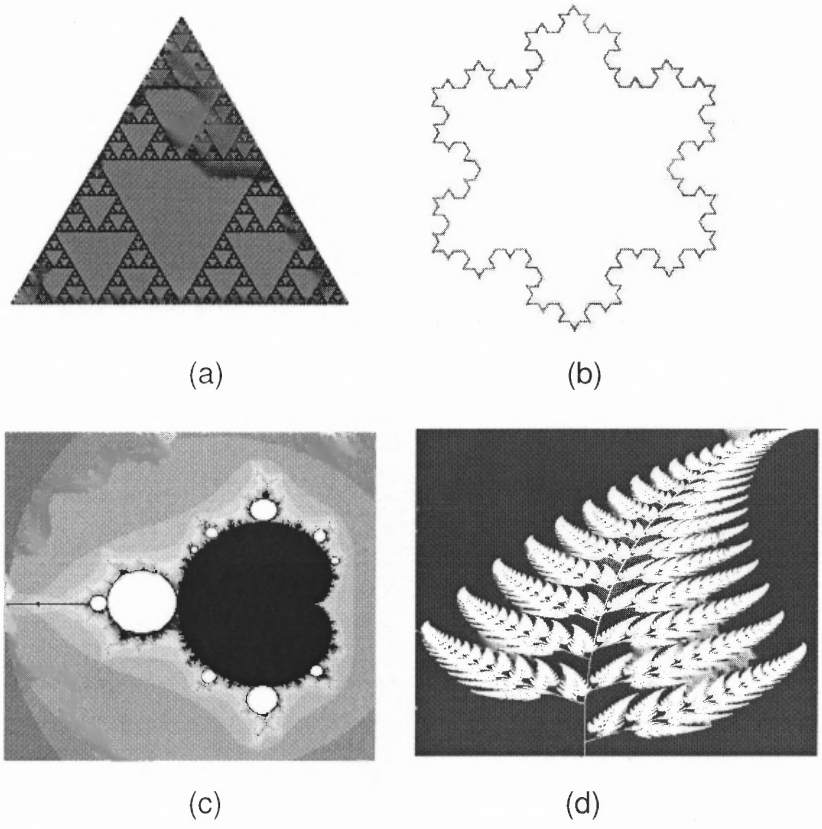


Figure C.1 (a) Sierpinski triangle; (b) Koch snowflake; (c) Mandelbrot set; (d) Fern.

Sierpinski triangle. Theoretically, the Sierpinski triangle is derived from an equilateral triangle ABC, by excluding its middle and recursively repeating this procedure for each of the resulting smaller triangles. The resulting set of points exhibits "holes" in any scale; moreover, each smaller triangle is a miniature replica of the whole triangle. In general, the characteristic of fractals is this self-similarity property: parts of the fractal are similar (exactly or statistically) to the whole fractal. The Sierpinski triangle gives an example of points which follow a highly non-uniform but deterministic distribution. There should be a way to describe it mathematically.

Often high-dimensional vector space suffer from large differences between their *embedding dimensions* and *fractal dimensions*. Embedding dimension (E) refers to

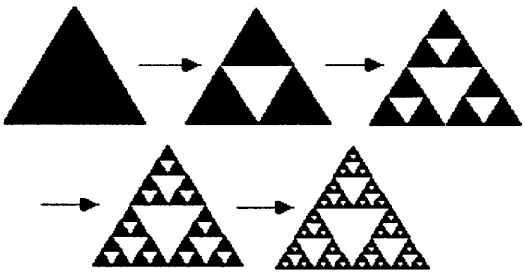


Figure C.2 Five steps in generating Sierpinski triangles.

the dimensionality of the original data space, i.e., number of attributes or features. Fractal dimension (d) is the *intrinsic dimension* of a dataset and is defined as the real dimensionality in which the points can be embedded, while preserving the distances among them [23]. For example, a line embedded in a 100-dimensional space has intrinsic dimension 2 and embedding dimension 100.

The *Hausdorff* fractal dimension or *box-counting* fractal dimension is the basic type of fractal dimension and it is defined as follows [67]: Divide the E-dimensional space into hyper-cubic grid cells of side r. Let $N(r)$ denote the number of cells that are penetrated by the fractal (i.e., contain 1 or more points of it). Then the (box-counting) fractal dimension d_0 of a fractal is defined as

$$d \equiv \lim_{r \to 0} \frac{logN(r)}{log(1/r)} \tag{C.1}$$

This definition is useful for mathematical fractals, that consist of infinite number of points. For a point-set that has the self-similarity property in the range of scales (r_1, r_2), its Hausdorff fractal dimension d_0 is measured as [28]

$$d_0 = -\frac{\partial log(N(r))}{\partial log(r)} = constant, \quad r_1 < r < r_2 \tag{C.2}$$

Another popular fractal dimension often used for query cost model is *Correlation* fractal dimension and is defined as

$$d_2 = \frac{\partial log(\sum_i {p_i}^2)}{\partial log(r)} = constant, \quad r_1 < r < r_2 \tag{C.3}$$

C.2 Faloutsos's Query Cost Models

According to the study in [28] and [11], the real-world datasets also behave like fractals, with linear box-count plots. Fractal dimensions can help to estimate query cost because the previous query models [29, 5] assume data distribution to be uniform and independent ($d = E$), which is not true for real-world datasets with fractal dimension much smaller than embedding dimension. They believe that the previous estimations of R-tree cost tend to be too pessimistic. As fractal dimension is utilized, Faloutsos et al. create a function to estimate the number of page accesses for range queries on the R-tree:

$$P_{all}(\vec{q}) = \sum_{j=0}^{h-1} \frac{M}{C_{eff}^{h-j}} \prod_{i=1}^{E} (\sigma_j + q_i) \tag{C.4}$$

where P_{all} is the average number of nodes accessed by a query of size q for each dimension, h is the height of the tree, σ_j is the side size of rectangles in j-th level and $\sigma_j = (\frac{C_{eff}^{h-j}}{M})^{1/d_0} \quad j = 0, \ ..., \ h-1$ and C_{eff} is defined as the effective capacity of the nodes of the R-tree as the average number of entries per node: $C_{eff} = C \times u$ (u is the average node utilization and C is the maximum number of rectangles per node).

This equation can only be used for $\mathcal{L}_\infty$ norm and the query is assumed uniformly and independently distributed. A formula is given in [47] to estimate the cost of nearest neighbor search for both $\mathcal{L}_\infty$ and $\mathcal{L}_2$ on R-tree using Correlation fractal dimension. Also, the query can have same distribution as the data itself.

However, for higher than three dimensions an accurate function for $\mathcal{L}_2$ is not available. Instead, the results of $\mathcal{L}_\infty$ norm are used to give an upper bound and lower bound of query cost for $\mathcal{L}_2$ norm.

Assume the biased model, square queries of radius ε. The average number of query-sensitive anchors (the centers of gravity of the query regions) of an MBR with side length l is :

$$an_{mbr}^{L_\infty}(l,\varepsilon) = (M-1)\cdot(l+2\varepsilon)^{d_2} + 1 \tag{C.5}$$

Therefore, the average number of accesses of R-tree pages needed to answer a k-NN query with $\mathcal{L}_2$ distance is estimated as

$$\sum_{j=0}^{h-1} \frac{an_{mbr}^{L_\infty}(\sigma_j, \frac{d_{nn}^{L_\infty}(k')}{\sqrt{E}})}{C_{eff}^{h-j}} \leq P_{all}^{L_2}(k) \leq \sum_{j=0}^{h-1} \frac{an_{mbr}^{L_\infty}(\sigma_j, d_{nn}^{L_\infty}(k'))}{C_{eff}^{h-j}} \tag{C.6}$$

where σ_j and C_{eff} are same as those in Equation C.4, $an_{mbr}^{L_\infty}(\sigma_j, \frac{d_{nn}^{L_\infty}(k')}{\sqrt{M}})$ denotes the average number of query-sensitive anchors of an MBR with side length σ_j. For range query, just modify $d_{nn}^{L_\infty}(k')$to ε for ε-range query as following.

$$\sum_{j=0}^{h-1} \frac{an_{mbr}^{L_\infty}(\sigma_j, \frac{\varepsilon}{\sqrt{E}})}{C_{eff}^{h-j}} \leq P_{all}^{L_2}(\varepsilon) \leq \sum_{j=0}^{h-1} \frac{an_{mbr}^{L_\infty}(\sigma_j, \varepsilon)}{C_{eff}^{h-j}} \tag{C.7}$$

C.3 Böhm's Query Cost Models

Based on the above results, Böhm generates a more general query cost model which can be used for both $\mathcal{L}_\infty$ and $\mathcal{L}_2$ distance metrics and, especially, for high-dimensional indexing (e.g., X-tree) in [16], also the query can be either uniformly and independently distributed or with the same distribution as the dataset itself.

Actually, in high-dimensional space, a *Minimum Bounding Rectangle* (MBR) is a *hyper-rectangle.* However, for a range query, if Euclidean distance is the distance metric that applies, the shape of the query range should be a hyper-sphere (see the example in Chapter 2). To estimate how many pages are being accessed, one has to know the number of pages intersect with the query, therefore it is important to

calculate the intersection of a hyper-rectangle and a hyper-sphere. The *Minkowski Sum* [43] is utilized in [16] to enlarge the page region so that if the original page touches any point of the query hyper-sphere, then the enlarged page touches the center point of the query sphere. Then the calculation of page volume in the previous models becomes the calculation of the hyper-volume of Minkowski enlargement.

Some effects of high-dimensionality are considered in that paper, including boundary effects and the large extension of query region. Boundary effects refer to the phenomenon occurring in high-dimensional data spaces that all data and query points are likely to be near by the boundary of the data space, while the large extension of query region refers to the phenomenon that when dimensionality is getting higher, the query radius for range query becomes so large that approaches the radius of the whole space, given that the query selectivity is same as that in lower-dimensionality space. The combination of the two effects leads to the observation that large part of a typical range or k-NN query sphere must be outside the boundary of the data space.

C.4 The Revised Query Model for Range Queries

Although Böhm's model is more complicated, the principle is actually same as that of Faloutsos's. In both models, query costs are proportional to $Hyper\_volume(l, \varepsilon)^{d_2/E}$, where $Hyper\_volume(l, \varepsilon)$ is the hyper-volumes of the *inflated MBR* described in [47] or the hyper-volumes of the *Minkowski Sum* in [16]. Therefore, the author chooses Faloutsos's cost model [47] as basic model and improves it to be able to handle datasets with any high dimensionality for range query (based on the equation C.7).

The equation with $\mathcal{L}_2$ distance metric is as follows:

$$P_{all}(\varepsilon) = \sum_{j=0}^{h-1} \frac{M \cdot Hyper\_volume(\sigma_j, \varepsilon)^{d_2/E}}{C_{eff}^{h-j}} \tag{C.8}$$

where

$$Hyper\_volume(l, \varepsilon) = \sum_{i=0}^{E} C_E^i \cdot V_i^R(l) \cdot V_{E-i}^S(\varepsilon)$$

$V_i^R(l)$ denotes i-d hyper-rectangle with side l and $V_i^S(l)$ denotes i-d hyper-sphere with radius ε, $V_0^R(l) = 1$, $V_0^S(\varepsilon) = 1$, $V_1^R(l) = l$, $V_1^S(\varepsilon) = 2\varepsilon$.

C.5 Experiments

The cost model of range query Equation C.8 is verified using a 16-dimensional time series dataset with 100,000 signals, which is generated by *Random-Walk* model, a synthetic time series generator obtained from Dr. Byoung-Kee Yi. The feature extraction method of [79] is applied to generate five feature datasets with dimensionality (E) equals to 2, 4, 6. 8 and 16. Then Equation C.8 is used to estimate the average number of page P_{all} accessed by a range query with radius ε for each dataset. All data are normalized to a unit hyper-cube.

The fractal dimension can be calculated in Linear time [74] [1]. The query costs are averaged over 1000 biased queries. DR-tree (2.5v) library developed at the University of Maryland with some modifications to handle $\mathcal{L}_2$-based search is used to create R-trees and process queries. The page size is set to 4096.

The results of estimations and experimental results are listed in Table C.1. The first column is the feature dimensionality and the next two columns show values of d_0 (Hausdorff fractal dimension) and d_2 (Correlation fractal dimension), and then comes the results of original equations Equation C.6 and the results of Equation C.8. The last two columns are experimental results without packing and with packing. The packing algorithm is from [49].

Compared to the experimental values, the estimates are much smaller and do not increase with the dimensionality as they should have been (like those of experimental

[1]The code used in this study is from Leejay Wu's web page:http: //www.andrew.cmu.edu/ lw2j/downloads.html

Table C.1 Results of Equation vs. Experiments ($M = 100,000$, $E = 16$, $\varepsilon = 0.1$)

E	d_0	d_2	Formula Results			Experimental Results	
			Lower B.	Upper B.	Exact	w/t Packing	w Packing
2	1.6907	1.9378	6.9	10.3	82.0	252.2	177.6
4	2.5575	3.2649	2.9	6.0	7.2	413.6	318.4
6	2.6153	3.9517	1.7	4.1	10.5	566.4	533.2
8	2.6861	4.606	2.4	5.3	6.1	796.5	739.7
16	2.5370	4.999	1.6	7.5	11.3	1910.3	1816.0

results). According to the equation C.8, the cost is proportional to $hyper - volume$ versus C_{eff} (without lost of generality here only number of leaf node accessed is considered). While, when dimensionality increases, the hyper-volume drops dramatically because l and ε are very small (much less than 1.0) due to the normalization, even though ε increases a little. Therefore, although C_{eff} also decreases, the cost still drops.

From Table C.1, it is also seen that for experimental results, the difference between packing or no packing is insignificant when dimensionality becomes higher. This means packing algorithm, at least the STR algorithm in [49] is not efficient for high-dimensional.

REFERENCES

[1] C. Aggarwal. On the effects of dimensionality reduction on high dimensional similarity search. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, pages 256–266, 2001.

[2] C. Aggarwal and P. Yu. Finding generalized projected clusters in high dimensional space. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 70–81, 2000.

[3] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *Proc. 4th Int'l Conf. on Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, 1993.

[4] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 94–105, 1998.

[5] W. Aref and H. Samet. Optimization strategies for spatial query processing. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, pages 81–90, 1991.

[6] M. Arya, W. F. Cody, C. Faloutsos, J. Richardson, and A. Toya. Qbism: A prototype 3-d medical image database system. *IEEE Data Eng. Bull.*, 16(1):38–42, 1993.

[7] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[8] S. D. Backer and P. Scheunders. A competitive elliptical clustering algorithm. *Pattern Recognition Letter*, 20(11-13), 1999.

[9] R. Bayer and E. McCreight. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1(3):173–189, 1972.

[10] N. Beckman, R. S. H. P. Kriegal, and B. Seeger. The R* tree: an efficient and robust access method for points and rectangles. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 322–331, 1990.

[11] A. Belussi and C. Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, pages 299–310, 1995.

[12] A. Belussi and C. Faloutsos. Self-spacial join selectivity estimation using fractal concepts. *ACM Transactions on Information Systems*, 16(2):161–201, 1998.

[13] S. Berchtold, D. Keim, and H. Kriegel. The X-tree: an index structure for high-dimensional data. In *Proc. 22nd Intl. Conf. on Very Large Databases (VLDB)*, pages 28–39, 1996.

[14] P. Berkhin. Survey of clustering data mining techniques. Technical Report, Accrue Software, 2002. `http://www.accrue.com/products/rp_cluster_review.pdf`, March 2nd, 2004.

[15] K. Beyer, R. R. J. Goldstein, and U. Shaft. When is "nearest neighbor" meaningful? In *Proc. on International Conference on Data Engineering (ICDE)*, pages 217–235, 1999.

[16] C. Böhm. A cost model for query processing in high-dimensional data space. *ACM Transactions on Database Systems (TODS)*, 25(2):129–178, 2000.

[17] C. Böhm and H. Kriegel. Efficient bulk loading of large high-dimensional indexes. In *Proc. Int'l Conf. on Data Warehousing and Knowledge Discovery*, pages 251–260, 1999.

[18] V. Castelli and L. Bergman, editors. *Image Databases: Search and Retrieval of Digital Imagery*. John Wiley and Sons, 2002.

[19] V. Castelli, A. Thomasian, and C. S. Li. CSVD: clustering and singular value decomposition for approximate similarity search in high dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(3):671–685, 2003.

[20] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series database. *ACM Transactions on Databse Systems (TODS)*, 27(2):188–228, 2002.

[21] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature space. In *Proc. on International Conference on Data Engineering (ICDE)*, pages 440–447, 1999.

[22] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional space. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, pages 89–100, 2000.

[23] E. Chevez, G. Navarro, R. Baeza-Yates, and J. Marroqun. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):271–321, 2001.

[24] P. Ciaccia, M. Patella, and P. Zezula. M-tree: an efficient access method for similarity search in metric spaces. In *Proc. 23nd Int'l Conf. on Very Large Data Bases (VLDB)*, pages 426–435, 1997.

[25] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[26] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd. Conf. ACM Special Interest Group on Knowledge Discovery in Data and Data Mining (SIGKDD)*, pages 226–231, 1996.

[27] C. Faloutsos. *Searching Multimedia Databases by Content.* Kluwer Academic Publishers, Boston, MA, 1996.

[28] C. Faloutsos and I. Kamel. Beyond uniformity and independence: Analysis of the R-tree using the concept of fractal dimension. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, 1994.

[29] C. Faloutsos, T. Sellis, and N. Roussopoulos. Analysis of object oriented spatial access methods. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 426–439, 1987.

[30] F. Farnstrom, J. Lewis, and C. Elkan. Scalability for clustering algorithms revisited. *SIGKDD Explorations Newsletter*, 2(1):51–57, 2000.

[31] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi. Approximate nearest neighbor searching in multimedia databases. In *Proc. on International Conference on Data Engineering (ICDE)*, pages 503–511, 2001.

[32] R. Finkel and J. Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

[33] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

[34] D. L. Gall. Mpeg: a video comparession standard for multimedia applications. *Communication of the ACM (CACM)*, 34(4):46–58, 1991.

[35] S. Guha, R. Rastogi, and K. Shim. CURE: an efficient clustering algorithm for large databases. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 73–84, 1998.

[36] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 47–57, 1984.

[37] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proc. 4th Symp. Advances in Spatial Databases*, pages 83–95, 1995.

[38] A. Jain, H. Lin, S. Pankanti, and R. Bolle. An identity-authentication system using fingerprints. *Proceedings of the IEEE*, 85(9):1365–1388, 1997.

[39] A. K. Jain and A. Vailaya. Image retrieval using color and shape. *Pattern Recognition*, 29(8):1233–1244, 1996.

[40] H. Jin, B. Ooi, H. Shen, and C. Yu. An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing. In *Proc. on International Conference on Data Engineering (ICDE)*, pages 87–100, 2003.

[41] I. T. Jolliffe. *Principal Component Analysis, 2nd. edition.* Springer-Verlag New York, Inc., 2002.

[42] N. Katayama and S. Satoh. The SR-tree: An index structure for high dimensional nearest neighbor queries. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 369–380, 1997.

[43] A. Kaul and M. O'Connor. Computing minkowski sums of regular polyhedra. Technical Report RC 18891 (82557) IBM T.J. Watson Research Center, 1992.

[44] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, 2001.

[45] B. Kim and S. Park. A fast k-nearest-neighbor finding algorithm based on the ordered partition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 8(6):761–766, 1986.

[46] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 289–300, May 1997.

[47] F. Korn, B. Pagel, and C. Faloutsos. On the"dimensionality curse" and the "self-similarity blessing". In *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, pages 96–111, 2001.

[48] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopaps. Fast nearest neighbor search in medical image databases. In *Proc. 22nd Int'l Conf. on Very Large Data Bases (VLDB)*, pages 215–226, 1996.

[49] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez. STR: A simple and efficient algorithm for r-tree packing. In *Proc. on International Conference on Data Engineering (ICDE)*, pages 597–605, 1997.

[50] C. Li, P. Yu, and V. Castelli. Hierarchy-scan: a hierarchical algorithm for similarity search of databases consisting of long sequences. In *Proc. on International Conference on Data Engineering (ICDE)*, pages 546–553, New Orleans, LA, 1996.

[51] Y. Li, A. Thomasian, and L. Zhang. An exact k-nearest neighbor search algorithm for CSVD. Technical Report ISL-2003-02, Integrated Systems Lab, Dept. of Computer Science, New Jersey Institute of Technology, 2003.

[52] Y. Li, A. Thomasian, and L. Zhang. Finding optimal subspace dimensionality for k-NN search in clustered datasets. Technical Report ISL-2004-01, Integrated Systems Lab, Dept. of Computer Science, New Jersey Institute of Technology, 2004.

[53] T. Lillesand and R. W. Kiefer. *Remote Sensing and Image Interpretation.* Wiley and Sons, New York, 4th edition, 2000.

[54] K. Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree: an index structure for high-dimensional data. *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, 3(4):517–542, 1994.

[55] Y. Linde, A. Buzo, and R. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications, com-28*, pages 84–95, 1980.

[56] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 18(8):837–842, 1996.

[57] H. Miller and J. Han, editors. *Geographic Data Mining and Knowledge Discovery.* Taylor and Francis, New York, NY, 2001.

[58] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Pektovic, P. Yanker, C. Faloutsos, and G. Taubin. The QBIC project: Querying images by content using color, texture,, and shape. In *Proc. SPIE Vol. 1908: Storage and Retrieval for Image and Video Databases*, pages 173–187, 1993.

[59] J. Nievergelt, H. Hinterberger, and K. Sevcik. The grid file: An adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems (TODS)*, 9(1):38–71, 1984.

[60] A. V. Oppenheim and R. W. Schafer. *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1975.

[61] B. Pagel, H. Six, H. Toben, and P. Widmayer. Towards an analysis of range query performance in spatial data structures. In *Proc. ACM Symposium on Principles of Database Systems (PODS)*, pages 214–221, 1993.

[62] E. Petrakis and C.Faloutsos. Similarity searching in medical image databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 3(9):435–447, 1997.

[63] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C, the Art of Scientific Computing.* Cambridge University Press, Cambridge, United Kingdom, 2nd edition, 1992.

[64] J. Richards. *Remote Sensing Digital Image Analysis, An Introduction.* Wiley and Sons, New York, 1993.

[65] J. T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 10–18, 1981.

[66] N. Roussopoulos, S. Kelley, and F. Vincent. Nearest neighbor queries. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 71–79, 1995.

[67] M. Schroeder. *Fractals, Chaos, Power Laws: Minutes From an Infinite Paradise.* W. H. Freeman and Company, New York, 1991.

[68] T. Seidl and H. P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 154–165, 1998.

[69] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+ tree: a dynamic index for multi-dimensional objects. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, pages 507–518, 1987.

[70] J. R. Smith. *Integrated Spatial and Feature Image System: Retrieval, Analysis and Compression.* Phd ttesis, Electrical Engineering Dept, Columbia University, 1997.

[71] K. Sung and T. Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 20(1):39–51, 1998.

[72] A. Thomasian, V. Castelli, and C. S. Li. RCSVD: Recursive clustering and singular value decomposition for approximate high-dimensionality indexing. In *Proc. Conf. on Information and Knowledge Management (CIKM)*, pages 267–272, 1998.

[73] A. Thomasian, Y. Li, and L. Zhang. Performance comparison of local dimensionality reduction methods. Technical Report ISL-2003-01, Integrated Systems Lab, Dept. of Computer Science, New Jersey Institute of Technology, 2003.

[74] C. Traina Jr., A. J. M. Traina, L. Wu, and C. Faloutsos. Fast feature selection using fractal dimension. In *XV Brazilian Symposium on Databases (SBBD)*, October 2000.

[75] W. Wang, J. Yang, and R. Muniz. STING: a statistical information grid approach to spatial data mining. In *Proc. 23rd Int'l Conf. on Very Large Data Bases (VLDB)*, pages 186–195, 1997.

[76] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 194–205, 1998.

[77] D. White and R. Jain. Similarity indexing with ss-trees. In *Proc. on 12th International Conference on Data Engineering (ICDE)*, pages 516–523, 1996.

[78] H. Williams and J. Zobel. Indexing nucleotide databases for fast query evaluation. In *Proc. 5th Int'l Conf. on Extending in Database Technology (EDBT)*, pages 275–288, 1996.

[79] B. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp norms. In *Proc. 26th Int'l Conf. on Very Large Data Bases (VLDB)*, pages 385–394, 2000.

[80] C. Yu, B. Ooi, K. Tan, and H. Jagadish. Indexing the distance: An efficient method to knn processing. In *Proc. Int'l Conf. on Very Large Data Bases (VLDB)*, pages 421–430, 2001.

[81] D. S. Zhang and G. Lu. Evaluation of similarity measurement for image retrieval. In *IEEE Int'l Conf.on Neural Networks and Signal Processing*, pages 928–931, 2003.

[82] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *Proc. Conf. ACM Special Interest Group on Management of Data (SIGMOD)*, pages 103–114, 1996.