EFFICIENT DATABASE SUPPORT FOR WWW IMAGE RETRIEVAL

By

Heng Tao Shen

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
AT
NATIONAL UNIVERSITY OF SINGAPORE
REPUBLIC OF SINGAPORE
JUNE 2003

NATIONAL UNIVERSITY OF SINGAPORE
DEPARTMENT OF
COMPUTER SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "**Efficient Database Support for WWW Image Retrieval**" by **Heng Tao Shen** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

Dated: June 2003

External Examiner: _____
Professor

Research Supervisor: _____
Professor Beng Chin Ooi

Examing Committee: _____
Professor

_____
Professor

To My Parents

Table of Contents

List of Tables

List of Figures

Acknowledgements

There are a number of people who guided and assisted me in one way or another to accomplish this research. First of all, I wish to thank Professor Beng Chin Ooi, my supervisor, for his bright guidance, insightful suggestions and constant support. During the past years, he built my confidence and shaped my research capability to stand higher. His guidance, trust and confidence on me are the keys for me to succeed in this research. Without him, I would not have been awarded for the Dean's Graduate Award in School of Computing, National University of Singapore.

Another important person for this research is Professor Kian-Lee Tan, who advised me in various ways to improve my research acumen. His comments on writing skills made me understand how to present a paper well. Moreover, his excellent edition has greatly polished this thesis's readability.

Next, I would like to thank Nick Koudas, from AT&T Shannon Laboratory USA, and H V Jagadish, Professor from University of Michigan Ann Arbor, for their discussion and cooperation in part of this research, especially on the hyper-dimensional indexing and multi-features indexing. They provided insightful suggestions and comments on the research proposals.

Working with my buddies, Shu Guang Wang, Bin Cu, Wee Siong Ng, and all other members in the Database groups, colored my research life.

Finally, but not the last, I would like to thank my beloved parents, for their endless love, forever.

Abstract

WWW is exploding and shaping the current research direction. To enhance the WWW page content, images are increasingly being embedded in HTML documents. Such documents over the WWW essentially provide a rich and interesting source of image collection from which users can query.

WWW images are described by both high-level feature - text, and low-level features - color, shape, and texture. Typically, each feature is represented as a high-dimensional feature vector. Unfortunately, most WWW image search engines fail to exploit image semantics and give rise to low precision. On the other hand, existing indexing techniques fail to provide more efficient retrieval than sequential scan as the dimensionality of image features reaches high due to the well-known 'dimensionality curse'. Moreover, the problem of indexing multiple image features is too hard to have been addressed.

In this thesis, we first propose an effective semantic-based WWW image retrieval system, and extend it with multiple visual features. To provide efficient database support, we then study the problem of high-dimensional indexing, from which we further address the problems of **hyper-dimensional**[1] indexing and multiple high-dimensional indexing.

To improve the retrieval accuracy of WWW images system, we present ICICLE (Image ChainNet & Incremental CLustering Engine), a prototype system that we have developed to effectively and efficiently retrieves WWW image by using the surrounding text, the high-level feature of images, to represent the semantics of images.

[1]The term hyper-dimensional is used to differentiate the problem we are addressing from the present norm of 30- to 50- (high) dimensional space

ICICLE has two distinguishing features. First, it employs a novel image representation model called Weight ChainNet to capture the semantics of the image content. Second, to search a large set of images quickly, we partition the images into clusters. ICICLE employs an incremental clustering mechanism, ICC (Incremental Clustering on ChainNet), that narrows the search space of the retrieval process to the relevant partitions. Moreover, ICC facilitates incremental updates and can adaptively adjust the number of clusters and cluster sizes. We conducted an extensive performance study to evaluate ICICLE. Our results show that ICICLE provides better precision and efficiency than existing techniques. To include image's low-level features, we extend ICICLE architecture to be adaptive for multiple features. Three novel indexing techniques are embedded in the extended ICICLE to speed up image searching.

To efficiently support image retrieval with high-dimensional feature, we present an adaptive Multi-level Mahalanobis-based Dimensionality Reduction (MMDR) technique to index the image databases in reduced much lower dimensional subspace. Our MMDR technique has four notable features compared to existing methods. First, it discovers elliptical clusters using only the low-dimensional subspaces to perform effective dimensionality reduction. Second, data points in the different axis systems are indexed using a single B^+-tree. Third, our technique is highly scalable in terms of data size and dimension. Finally, it is also dynamic and adaptive to insertions. An extensive performance study was conducted, and the results show that our technique not only achieves higher precision, but also enables queries to be processed efficiently.

However, the image features, such as texture and shape, can reach up to hundreds or more. Such hyper-dimensional features pose significant problems to existing high-dimensional indexing techniques. To support efficient querying and retrieval on hyper-dimensional databases, we propose a methodology called Local Digital Coding (LDC) which can support K-Nearest Neighbors (KNN) queries on hyper-dimensional databases and yet co-exist with ubiquitous indices, such as B^+-trees. LDC extracts a simple bitmap representation called Digital Code(DC) for each point (or feature vector) in its nature space. Pruning during KNN search is performed by dynamically selecting only a subset of the bits from the DC based on which subsequent comparisons are performed. In doing so, expensive operations involved in computing L-norm

distance functions between hyper-dimensional data can be avoided. Extensive experiments are conducted to show that our methodology offers significant performance advantages over other existing indexing methods on hyper-dimensional datasets.

To speed up retrieval with multiple high-dimensional image features, we devise a novel image representation that compactly captures f features into two vector components: the first component is an f-dimensional vector where the ith feature is transformed into a value in a single dimension space, and the second component is a bit sequence, with two bits per dimension, obtained by analyzing each feature's distance histogram. This representation leads to a single two-level index structure where the first tier indexes the first component using a standard multi-dimensional index structure such as an R-tree, and the second level is a compact list of bit sequences accessible from the leaf node entries of the first level. The proposed two-tier structure automatically brings about dimensionality reduction. It also permits features to be weighted on a per query basis, so that a single index structure can support a variety of different similarity measures. In particular, it can also support queries that do not specify all features. We also propose an efficient algorithm for processing KNN queries. Our extensive experiments indicate that the proposed index structure offers significant performance advantages over sequential scan and retrieval methods using single and multiple existing indexes.

In short, ICICLE [50, 49, 48, 40] is a more effective and efficient WWW image retrieval system. The proposed indexing techniques MMDR [31] for high-dimensional feature indexing, LDC [33] for hyper-dimensional feature indexing, and single two-tier index structure [30] for multi-features indexing provide strongly efficient database support for extended ICICLE.

Chapter 1

Introduction

Modern advances in image processing technology have made the image retrieval an active research topic. As the Internet bandwidth increases rapidly and hardware technologies develop quickly, free publishing of images in World Wide Web (WWW) pages have become very prevalent. However, the semantics of WWW images has never been fully explored to support effective retrieval. Beside the *effectiveness* issue, the other essential issue for an image retrieval system is its *efficiency* to support fast retrieval.

Database management systems are standard tools for manipulating large database. To speed up access in a database, data organization structures, known as indexes, are usually deployed. It is known that indexes are the primary means for speeding up data retrieval and designing effective indexing structures are one of the most important research areas in the database literature. Images are described by their features, such as color, shape, texture, and text. Each feature of an image is typically transformed into a high-dimensional (up to hundreds or more) point after some feature transformation techniques. The state-of-art indexing methods have been shown not to be scalable to high-dimensional spaces due to the well-known 'dimensionality curse'. An image is typically described by multiple features. Thus image databases are in multiple high-dimensional spaces. Unfortunately, the problem of indexing multiple

high-dimensional spaces is seldom addressed.

In this thesis, we propose an effective semantic-based WWW image retrieval system, and study the problem of indexing image database to provide efficient support.

1.1 Content-Based Image Retrieval (CBIR)

1.1.1 What is CBIR?

The use of images in human communication can be traced back to thousands of years. Our cave-dwelling ancestors painted pictures on the walls, and used maps to convey needed information. As time goes on, images now play a crucial role in fields as diverse as medicine, journalism, advertising, design, education, entertainment, and so on. As the volume of images is increased rapidly, the need for effective and efficient retrieval of relevant images from a large and varied collection is recognized. As a result, image retrieval has been an active research topic and has gained steady momentum as a result of the dramatic increase in the volume of images. More recently, the term - Content-Based Image Retrieval (CBIR) has been widely used to describe the process of retrieving desired images from a large collection on the basis of features which refer to the most common low-level/visual features: color, shape and texture. In the literature, many CBIR systems, such as [44, 53, 39, 46] etc, have been proposed. However, retrieval of images by manually-assigned keywords is definitely not CBIR as the term is generally understood - even if the keywords describe the image content.

CBIR operates on a totally different principle from keyword indexing. Retrieval of images are based on the similarity of images with respect to a given image as a query. Image features are usually represented as high-dimensional feature vectors (or points), i.e., each feature vector contains D values, which corresponds to coordinates in a D-dimensional space. The similarity between images are measured by some

distance functions i.e., comparing the feature vectors of the images. The result of this process is a quantified similarity score that measures the visual distance between the two images represented by the feature vectors. Queries are expressed through visual examples, which can either be formulated by users or selected from randomly generated image sets. If multiple features are involved, the similarity from each feature is integrated to get an overall score. And feature characteristics of the query image can be specified and weighted against each other. Searching queries returns a ranked result set instead of exact matches. Besides, the user mostly wants to see only the K top-ranked images. Low-level/visual features characterizing image content, such as color, shape and texture, are computed for both stored and query images, and used to identify the top K most similar images.

1.1.2 Problems of CBIR

However, the techniques for effective and efficient image retrieval have not kept pace with the technology of image production and speed of image increase.The effectiveness of all current CBIR systems is inherently limited by the fact that they can operate only at the low-level feature level. High-level/semantic features such as the type of object and the event presented in the image are harder to extract, though this remains an active research topic. None of them can search effectively for, say, a photo of 'Bill Clinton'. There is evidence that combining low-level image features with high-level features (i.e., text description) can overcome some of these problems. Some existing systems combined keywords and low-level features [64, 65, 5, 12, 37, 21, 54, 52] in order to improve the accuracy. However, it is not practical to manually enter the keywords for a large collection of images. Furthermore, too few key words may not be enough to describe an image.

On the other hand, the efficiency of all current CBIR systems is limited by the

long retrieval time for large collections. As the number of images reaches millions or billions, scanning every stored image for matching is definitely not desirable. Hence, while people in image retrieval research area focus more on effectiveness issue, image database application has also attracted database researchers to design effective indexing methods to support efficient retrieval. The problem of finding the K top-ranked images is equivalent to K-Nearest Neighbors (KNN) problem that has been addressed by the database community. Due to the large quantity of images and high dimensionality of image features, efficient indexing methods are necessary to speed up the searching and retrieval. Indexing high-dimensional data has been an active area of research for a long time and many indexing techniques have been proposed, including early works on multi-dimensional indexing structures (less than ten) [22] and recent indexing structures for high-dimensional data (less than hundred) [8]. However, the performance of these indexes degrades rapidly with increasing dimensionality due to the known 'dimensionality curse'. Moreover, image features usually have hundreds or more dimensionality. Existing structures are not scalable for such high-dimensionality [9].

Hyper-dimensional databases are databases which contain hundreds or even thousands of dimensions. Apart from image database, recent advances in several research fields including other multimedia types, bioinformatics, data mining on audio and text, as well as networking, have resulted in such databases which pose significant challenges to existing high-dimensional indexing techniques, that are usually capable of handling databases (commonly) up to tens of dimensions. The problem of indexing and searching in a hyper-dimensional database is a challenging one, due to three main reasons:

- First, according to several studies (e.g., [9]), the expected minimal distance between any two points in a hyper-dimensional space is very large (becoming

larger with increasing dimensionality) while the difference between the minimal and maximal distance to a point is expected to be small (becoming smaller with increasing dimensionality). These two characteristics of a hyper-dimensional space mean that the search radius for a k-nearest neighbor query is expected to be large. This in turn results in a large number of "false positives" since most points are expected to have almost equal distance to the query point. This phenomenon leads to significant deterioration of the query performance in most existing indexing methods.

- Second, due to the extremely high dimensionality, the fanout for most indexes built on a hyper-dimensional space is typically very small, resulting in an increase in the height of the indexes (e.g., in a 200 dimensional space, we can't expect more that ten entries in an 8K page if 4 bytes are needed for each dimension).

- Finally, the computation of the distance (e.g., Euclidean distance) between two points in a hyper-dimensional space, becomes processor intensive as the dimensionality increases. This implies that the processor time is expected to become a significant portion of the overall query response time for a hyper-dimensional database. Proposed techniques for optimizing the performance of most indexing techniques do not take this into consideration.

Another interesting aspect for image databases is that images are typically described by multiple features (or multi-feature). For example, an image may be described by a 64-dimensional color, a 64-dimensional shape, and a 64-dimensional texture. This phenomenon also occurs in many other emerging database applications, such as exploratory data analysis, market basket applications, bioinformatics and time-series matching. A query consisting of multiple features are referred as

multi-feature or complex query. To support multi-feature queries, we can build a high dimensional index on the feature space obtained from all dimensions of the multiple features. In the above image example, this corresponds to an 192-dimensional feature space. Unfortunately, such an approach is not likely to be effective because of the high dimension. Moreover, existing high-dimensional indexing techniques typically treat all the different dimensions homogenously. An alternative approach is to build one index for each feature. In this case, multi-feature queries are evaluated by integrating results from each index to get the final rank-ordered results. However, combining answers from multiple indexes for ranked queries may require examining a large portion of each index.

1.1.3 Searching Images from WWW

With the increase in Internet bandwidth and CPU processing speed, the use of images in WWW pages has become very prevalent. Images are used to enhance description of content, to capture attention of readers and to reduce the textual content of a WWW page. An image is worth 1,000 words. Images have become an indispensable component of WWW pages today. Hence WWW provides an interesting and super-large special pool of images, which consists of both high-level and low-level features. This pool of WWW images becomes a very rich source from which users can obtain interesting images. However, as the web crawler keeps crawling, the growing number of images embedded in WWW pages makes the WWW a gigantic image database. To retrieve relevant images from this collection poses two challenges to the research community. First, as an improvement of CBIR, more semantic-based effective (measured in terms of recall and precision) method should be designed. Second, the exponential growth rate of images in WWW would eventually, if not already, render any existing techniques ineffective and inefficient.

1.2 The Objectives and Contributions

In this thesis, we present our solutions to address the issues of effectiveness and efficiency for WWW image retrieval. To tackle the effectiveness problem , we employ a novel scheme to capture the semantics of an image within a HTML document. To speed up the searching process, two research approaches are considered: clustering and indexing. One cluster clustering method and three novel indexing methods are proposed. Extensive performance study are conducted to demonstrate the superiority of the proposed methods.

1.2.1 Semantic-based WWW Image Retrieval

To capture the semantics of WWW images, we propose a novel image representation model called weight ChainNet. This is based on the observation that an image in a Web page is typically semantically related to its surrounding texts, with the exception of functional images (such as new symbol and under construction symbol). These surrounding texts are used to illustrate some particular semantics of the image content, i.e., what objects are in the image, what is happening and where the place is. In particular, in a HTML document, certain components are expected to provide more semantic information than other portion of the text. These include the caption of the image, its title and the title of the document. Weight ChainNet is based on Lexical Chain obtained from an image's nearby text, where Lexical Chain is defined as a sentence of words. A new formula, called list space model, for computing semantic similarities is also introduced. To further improve the retrieval effectiveness, we also propose two relevance feedback mechanisms.

To overcome the efficiency problem for our semantic-based retrieval, we propose that the database be split into multiple smaller partitions based on the semantic

representation model mentioned above. To this end, we propose a novel clustering scheme, called ICC (Incremental Clustering on ChainNet) that clusters images with similar semantics into the same partition. ICC facilitates incremental updates. In this way, the newly added data are inserted into the relevant partitions or a "noise" partition. In addition, ICC can dynamically adjust the number of partitions and the partition size by splitting larger partitions or merging small partitions. ICC is supported by two important mechanisms. First, it employs a hierarchical tree structure, Hierarchical-ChainNet Summarization Tree (denoted HC-ST), whose leaf nodes represent summary information of clusters (one leaf node per cluster), and whose internal nodes contain summary data on their children nodes. Second, the summary data at internal nodes are obtained using a two-step novel scheme, called Vertical and Pyramidal Summarization Tree (VP-ST). Given a query image, we first locate the partitions that contain images that are relevant to it. This is done by comparing its ChainNet with that of the summary ChainNet at internal nodes. Finally, the relevant partitions are examined.

We implemented a prototype WWW image retrieval system, called ICICLE (Image ChainNet & Incremental Clustering Engine) that employed the proposed mechanisms. And the system is further extended to take visual features into account, i.e., integrate with content-based retrieval. To provide efficient database support for the extended ICICLE, we propose three indexing techniques to tackle the problem of high-dimensional indexing and multi-feature indexing.

1.2.2 High-dimensional Indexing

To minimizing the effect of 'dimensionality curse', one approach is to reduce the number of dimensions of the high-dimensional data before indexing on the reduced dimension [42, 13]. Data is first transformed into a much lower dimensional space

using dimensionality reduction methods and then an index is built on it. Transforming data from a high-dimensional space to a lower dimensional space without losing critical information is not a trivial task. We propose a dimensionality reduction technique called Multi-level Mahalanobis-based Dimensionality Reduction (MMDR) for indexing based on the following two observations. First, elliptical shaped (correlated) clusters are more suitable for dimensionality reduction than spherical shaped clusters. Second, we observe that certain level of the lower dimensional subspaces may contain sufficient information for correlated cluster discovery in the high-dimensional space. In the MMDR, Principal Component Analysis(PCA) [32] is employed to find the lower dimensions for dimension reduction. Most of the information in the original space can be condensed into a few dimensions along which the variances in the data distribution are the largest. We make use of the *Mahalanobis distance* (MahaDist) in our approach instead of the standard well-known L-norm distance functions. Mahalanobis distance could be applied to find ellipsoidal correlated data, by taking local elongation into account. Based on multi-level low-dimensional projections produced by PCA and the Mahalanobis distance function, the MMDR can quickly identify highly correlated elliptical clusters. After the dimensionality reduction, each cluster of data is in a different axis system. Instead of creating one index for each cluster, we build one index for all the clusters for KNN queries. We extend a recently proposed B^+-tree based index - iDistance[61, 62], to index the data projections from the different reduced-dimensionality spaces. The extended iDistance allows us to index data points from different axis systems in a single index efficiently.

1.2.3 Hyper-dimensional Indexing

To enable searching in hyper-dimensional space, we propose an effective methodology called Local Digital Coding (LDC) for finding KNN in a hyper-dimensional space.

LDC is developed to address the problems mentioned above and provide a substantial reduction on both I/O and processor time when searching on hyper-dimensional datasets consisting of hundreds of dimensions. It is compatible with ubiquitous indices, such as B^+-trees and thus can be easily deployed. Given a cluster of points in a high-dimensional data space, LDC transforms each point into a bitmap which we refer to as the point's Digital Code (DC). Each dimension of the point is represented by a single bit in its DC. The DC of a point is generated by comparing the coordinates of the point with the coordinates of the cluster center the point belongs to. A bit is set to 1, if the value of the dimension it corresponds to, is larger than the value of the corresponding dimension of the cluster center, and 0 otherwise. Since there is a bit in the DC for each dimension, indexing a D-dimensional space will result in DCs with D bits. The data points in a cluster can thus be separated into 2^D partitions with points in each partition sharing the same DC. Based on LDC, we propose a novel searching algorithm, called *Searching on-the-fly by PArtial-distance* (SPA). Given the DCs of both the query point and a partition, SPA dynamically selects a *subset* from the DCs (say n bits) to perform matching. A partition is pruned off if the number of matching bits in the two DCs is less than m bits. The intuition behind such an approach is that the points in the pruned partition are on different sides of some cutting planes with respect to the query point and thus are too far away to be in the answer set.

1.2.4 Multi-features Indexing

To support multi-feature queries, we devise a novel representation that compactly captures f multi-dimensional features into two vector components. The first component is an f-dimensional vector obtained by transforming each of the f features into a value in a single dimension space. The second component is a bit sequence of size $2\sum_{i=1}^{f} d_i$ where d_i is the number of dimensions of the ith feature, i.e., each dimension

contributes two bits. The bits are set by analyzing each feature's distance histogram. This representation leads to a two-level index structure where the first tier indexes the first component using a standard multi-dimensional index structure such as an R-tree, and the second level is a compact list of bit sequences accessible from the leaf node entries of the first level. Our technique results in more effective indexing, as we experimentally demonstrate, for several reasons. First, high-dimensional indexing is hard, and most systems attempt to reduce dimensionality to the extent possible. Our two level decomposition automatically brings about this dimensionality reduction. Second, explicit identification of semantically meaningful features makes it easy to weight these features as desired, on a per query basis. For example, a query that cares only about color and shape (ignoring texture) as well as a query that cares about all four features can both be supported using one single index on image objects in our database. Third, high-dimensional indexing techniques often use a low-dimension projection for indexing [7, 62]. These techniques assume geometric homogeneity – all dimensions are considered equivalent – an assumption that is valid only within the dimensional attributes of a single feature. Our two-level decomposition permits these powerful reduction techniques to be applied one feature at a time. We also propose a novel KNN query searching algorithm called *Adaptive Searching by Aggressive Partial-distance* (ASAP) that iteratively prunes the search space aggressively based on the most critical dimensions of highly selective features.

Our extensive experiments show that the above methods improve the existing ones significantly and provide the efficient database support for the proposed effective WWW image retrieval system.

1.3 Organization of the Thesis

The organization of the rest of the thesis goes as follows:

In Chapter 2, we review an extensive related work in image retrieval literature. From the point of effectiveness, we review the existing image retrieval systems. On the other hand, from the point of efficiency, we review the existing indexing techniques which support fast retrieval.

In Chapter 3, we present the effective semantic-based WWW image retrieval system called ICICLE and its extension to adapt multiple features. In the next three chapters, we focus on the efficient database support on the image retrieval.

In Chapter 4, we propose a novel high-dimensional indexing technique called Multi-level Mahalanobis-based Dimensionality Reduction (MMDR) to effectively reduce the dimensionality of the original data space by adaptively identifying the correlation among the dimensions, then an index is built on the reduced subspaces.

In Chapter 5, we introduce a new methodology called Local Digital Coding (LDC) to support efficient querying and retrieval on hyper-dimensional space. LDC extracts a simple bitmap representation called Digital Code (DC) for each point in the database.

In Chapter 6, we devise a novel image representation that compactly captures f features into two vector components: the first component is an f-dimensional vector where the ith feature is transformed into a value in a single dimension space, and the second component is a bit sequence, with two bits per dimension, obtained by analyzing each feature's distance histogram. This representation leads to a two-level index structure to support efficient retrieval on multiple feature spaces.

Chapter 7 concludes this thesis with some discussion on future work.

Chapter 2

Related Work

2.1 Introduction

A number of image retrieval systems have been proposed in image retrieval literature. However, most of them are content-based and not for WWW images. The advances in database management have enabled fast access to image databases. In practice, indexing techniques have to be employed to index large database so as to meet the performance requirement. Many indexes have been proposed for high-dimensional (up to tens) databases in database literature. They are designed for typical operations of range search and KNN search. However, image features can be hundreds of dimensions or more. Recent research has shown that the performance of existing indexes deteriorates quickly as dimensionality increases and turns to be worse than sequential scan when dimensionality reaches few tens only.

In this chapter, we shall first review the existing work on image retrieval systems, followed by existing work on high-dimensional indexing techniques to support efficient retrieval. Finally, research efforts on indexing multiple features will be surveyed.

2.2 Image Retrieval Systems

With the increasing need in WWW image retrieval, many recent WWW image search engines have been developed in last decade. However, most of the existing image retrieval systems cannot adequately address the issue of effectiveness and efficiency. Text-based systems use keywords or free text description of images supplied by the authors as the basis for retrieval. These systems can be adopted for WWW images since the textual content of the HTML page in which the image is embedded provides the free text description. However, the entirety of the textual content does not represent the semantics of the image adequately for them to be useful in retrieving the images. In other words, while the textual content may contain information that captures the semantics of the embedded image, it also contains other descriptions that are not relevant to the image. These "noises" may lead to poor retrieval performance if query contains some of these noises. Many first generation of WWW text engines, like Lycos and Alta Vista, extracted keywords using standard algorithm that consider keyword placement, frequency, etc. They do not require solving the image's semantic structure indicated by image's surrounding text for better image understanding.

Typically, [51] conducted experiments on using semantic distances between words in image caption retrieval. They calculated word similarity between related words in a thesaurus. Similarity between words is used to identify if two images are relevant. Meanwhile, only image caption are involved for identification. More recently, [26] considered hyperlinks for WWW-based image collections. In [26], an image's content is given by the combined content of the text nodes. An image's set of text nodes include textural content (e.g., caption) obtained from the document in which it is embedded, as well as those obtained from its neighboring pages (those pages that are reached by a single hyperlink from the embedded page). This model was further

extended to take into account not only the textual content of the immediate neighbors of an image, but also all nodes that can be reached from the image by following at most two hyperlinks (a two-step link), thus considering more information about an image node. However, there are no explicit image/query semantics considered. The inner semantic relationship within a text node was lost based on this model. Moreover, while keeping more information is desirable, the approach extracted too much unrelated information, as relatively low precision can indicate. For example, an image's own caption usually describes its content, but its neighboring pages' image captions do not reflect the same content. In addition, the similarity measure did not take into account any semantic structure. Such a similarity measure may not be good enough to show the real semantic similarity between an image and a query. Relevance feedback (RF) is a very important way to improve the accuracy. The system refines the query by using feedback information from users to improve subsequent retrieval. The use of relevance feedback using multiple attributes of color has been investigated in [16]. Their results showed significant improvement in retrieval effectiveness by applying RF mechanisms.

On the other hand, content-based image retrieval systems such as [44, 53, 39, 46] etc, capture the visual content of an image (such as color, texture and shape) as its semantics and use these features as the basis for similarity matching. Unfortunately, retrieval by content is still far from perfect and their results are not reliable. First, their effectiveness depends on how precise the user specifies the query. Second, they are at very low performance levels as they cannot capture the more useful image semantics, like object, event, and relationship. Finally, they do not scale well. For the WWW image database, content-based image retrieval systems are not reliable, since low-level visual features cannot represent the high-level semantics of WWW images.

A combination of textual and visual features has been used in integrated image searching, such as [64, 65, 5, 12, 37, 21, 54, 52] etc. Works in above engines uses image features and associated text for automatic indexing of images. However, the key issue is how to obtain the high-level semantic features. Unfortunately, the image's surrounding descriptive texts are not well identified, but used to extract keywords for words matching purpose only. The internal semantic relationships among words are not remained anymore, which leads to poor precision in text-based searching component. Thus the overall performance is not very satisfied, as content-based systems are still in performance low level.

In terms of efficiency, most of the existing works employ indexing methods such as R-trees and its variants [41], the signature file [18] and hashing technique [11] to speed up the retrieval process for image database. While efficient for up to tens of dimensional databases, recent research [59] proved that these methods are not expected to scale well with very large image collections in high dimensions. Another direction to improve the efficiency of the system is to cluster the image collection into partitions. Most of the existing clustering schemes, however, are designed for static databases. Existing static clustering schemes, such as [43, 3, 4, 29, 47, 63, 24, 27] that have to perform the clustering from scratch should there be any new data to be added. Clearly, because the WWW image database keeps updating over time, they are not suitable for such kind of database. Few incremental clustering methods [15, 19] have been also proposed. However, they impose a fixed number of clusters as a constraint on the solution. Next, we review existing works on high-dimensional indexes which serve the basis for our design of scalable indexing methods.

2.3 High-dimensional Indexing

Indexing techniques have been the focus of extensive research both in low [22] as well as high-dimensional databases [8]. With the demand for even higher-dimensional databases, consisting of hundreds of or more dimensions, earlier high-dimensional indexes face significant challenges. Indexing techniques have been designed typically for 30-50 dimensions, and fail to improve the performance of sequential scan [59] due to the known "dimensionality curse". To tackle this phenomenon recent proposals adopt one of the three approaches: (1) Dimensionality reduction, (2) Data approximation, and (3) One dimensional Transformation.

2.3.1 Dimensionality Reduction

Dimensionality reduction methods [13] map the high dimensional space into a low dimensional space which can be indexed efficiently using existing multi-dimensional indexing techniques. The main idea is to condense the original space into a few dimensions along which the information is maximized. In dimension reduction for indexing, [13] proposed two strategies. In the first strategy, called the *Global Dimensionality Reduction* (GDR), all the data is reduced as a whole down to a suitable dimension on which search time and access costs are optimized. This strategy is unable to handle datasets that are not globally correlated. The other strategy, called the *Local Dimensionality Reduction* (LDR), divides the whole dataset into separate clusters based on correlation of the data and then indexes each cluster separately. Unfortunately, the LDR is not able to detect all the correlated clusters effectively, because it does not consider correlation nor dependency between the dimensions. Such methods report approximate nearest neighbors however, since dimensionality reduction incurs information loss.

To find meaningful clusters, clustering algorithms have been studied recently in the domain of data mining and pattern discrimination. Methods proposed for high-dimensional data clustering are related to our work. PROCLUS [3] clusters the data based on the correlation among the data along certain original dimensions. OptGrid [28] finds clusters in a high-dimensional space by projecting the data onto each axis and partitioning the data by using cutting planes at low-density points. Wavelet transform [56] and discrete cosine transform [34] based techniques rely on the partitioning of the data space into grids similar to OptGrid. These approaches do not work well when well-separated clusters in the actual space overlap after they are projected onto certain axis.

[2] presents various results of qualitative behaviors of L-norm distance matrices for measuring the proximity in high-dimensional spaces, and examines the meaningfulness of similarity in such spaces. They show that the clustering quality and answer sets vary from one distance metric to another. Beside L-norm distance functions, Mahalanobis distance has been used in face detection to discover actual non-isotropic face patterns among thousands of face images using a k-means like algorithm called the elliptical k-means method [55]. It is a nested loop algorithm, where the inner loop is to perform k-means using Mahalanobis distance and the outer loop is to recompute the covariance matrix of each cluster. Both loops stop when there is no change to the cluster membership. Such method is too expensive to be used for large high-dimensional image database, leaving optimization issues to be addressed.

2.3.2 Data Approximation

Representations of the original data points using smaller, approximate representations have also been proposed, as a means of aiding high dimensional indexing and searching. Such proposals include, the VA-file [59], the IQ-tree [6] and the A-tree [45].

The VA-file (Vector Approximation file) represents the original data points by much smaller vectors. The VA-file [59] employs a bit representation of the feature vector and has been shown to be superior to sequential scan in a uniformly distributed feature space. The main drawback of the VA-file however, is that it defaults in assessing the full distance between the approximate vectors, which imposes a significant overhead, especially if the underlying dimensionality is very large. Moreover, the VA-file does not adapt gracefully to highly skewed data. The IQ-tree was proposed recently. It maintains a flat directory which contains the minimum bounding rectangles of the approximate data representations. The basic idea of the A-tree is the introduction of virtual bounding rectangles (VBRs) which contain and approximate MBRs or data objects. VBRs can be represented quite compactly and thus affect the tree configuration both quantitatively and qualitatively.Each A-tree node contains an MBR and its children VBRs. Therefore, by fetching an A-tree node, information on the exact position of a parent MBR and the approximate position of its children can be obtained.

2.3.3 One Dimensional Transformations

One dimensional transformations provide another direction for high-dimensional indexing. Such techniques include the Pyramid technique[7]. iMinMax[42] and iDistance[62]. The Pyramid technique[7] divides the D-dimensional data space into 2D pyramids and then cuts each pyramid into slices each of which forms a data page. It provides a mapping from D-dimensional space to single-dimensional space. The iMinMax[42] transforms a high-dimensional point into either maximum or minimum of values among the various dimensions of the point. iDistance[62] transforms a high-dimensional point into a single-dimensional distance value with reference to its corresponding reference

point. They suffer however, from the fact that any meaningful search operation involves assessing distances between the full high dimensional representation of the data points; thus, pruning during search becomes problematic as the dimensionality increases.

Other techniques utilizing approximate data representations, such as hash-based method [23], return approximate, as opposed to exact, results on high-dimensional searches.

2.4 Multiple Feature Indexing

Little work has been reported on the problem of indexing multiple features, each of which is high-dimensional. Most existing image retrieval systems employ the method of multiple indices, i.e., building one index structure for each individual feature. To search for the relevant images from database, query processing has to be utilized among all the indices. Such operation is so called 'multi-feature query processing' [25, 20]. The major challenge here is to optimally combine the scores from all features in order to minimize the access cost. [25] proposed a method so called 'Quick-Combine' to combine multi-feature queries. It introduces an improved termination condition in tuned combination with a heuristic control flow adopting itself narrowly to the particular score distribution. KNNs can then be computed and output incrementally. [20] analyzes a simple and elegant algorithm so called 'the Threshold Algorithm' which is optimal in a much stronger sense.

The exception is the work by Ngu et al [38] that constructs a single M-tree [17] index for all the features. For efficient indexing, its method incorporates both the Principle Component Analysis and non-linear neural network techniques to reduce the dimensions of feature vectors so that an optimized access method can be applied.

To incorporate human visual perception into our system, experiments that involved a number of subjects classifying images into different classes for neural network training were also conducted. However, this method may not be practical for real usage. First, its neural network training process is tedious and undesirable for very large datasets. Second, existing indexing structures, such as M-tree, are known to degrade in performance for dimensionality larger than 20 [59]. This may result in significant information lost that may affect retrieval effectiveness.

Chapter 3

Semantic-based Retrieval for WWW Images

3.1 Introduction

With the increase in Internet bandwidth and CPU processing speed, the use of images in WWW pages has become very prevalent. Images are used to enhance description of content, to capture attention of readers and to reduce the textual content of a WWW page. An image is worth 1000 words. Images have become an indispensable component of WWW pages today. This pool of WWW images becomes a very rich source from which users can obtain interesting images. However, as the web crawler keeps crawling, the growing number of images embedded in WWW pages makes the WWW a gigantic image database. To retrieve relevant images from this collection poses two challenges to the research community. First, more semantic-based effective (measured in terms of recall and precision) method should be designed. Second, the exponential image growth rate would eventually, if not already, render existing techniques inefficient.

In this chapter, we present our solutions to address the issues of effectiveness and efficiency for semantic-based image retrieval, and extend it to take low-level features into consideration. To tackle the effectiveness problem, we employ a novel scheme

to capture the semantics of an image within a HTML document. This is based on the observation that an image in a Web page is typically semantically related to its surrounding texts, with the exception of functional images (such as new symbol and under construction symbol). These surrounding texts are used to illustrate some particular semantics of the image content, i.e. what objects are in the image, what is happening and where the place is. In particular, in a HTML document, certain components are expected to provide more semantic information than other portion of the text. These include the caption of the image, its title and the title of the document. We propose a novel image representation model called weight ChainNet. Weight ChainNet is based on lexical chain obtained from an image's nearby text. A new formula, called list space model, for computing semantic similarities is also introduced. To further improve the retrieval effectiveness, we also propose two relevance feedback mechanisms.

To speed up the searching process, we propose that the database be split into multiple smaller partitions based on the semantic representation model mentioned above. To this end, we propose a novel clustering scheme, called ICC (Incremental Clustering on ChainNet) that clusters images with similar semantics into the same partition. ICC facilitates incremental updates. In this way, the newly added data are inserted into the relevant partitions or a "noise" partition. In addition, ICC can dynamically adjust the number of partitions and the partition size by splitting larger partitions or merging small partitions. ICC is supported by two important mechanisms. First, it employs a hierarchical tree structure, Hierarchical-ChainNet Summarization Tree (denoted HC-ST), whose leaf nodes represent summary information of clusters (one leaf node per cluster), and whose internal nodes contain summary data on their children nodes. Second, the summary data at internal nodes are obtained using a two-step novel scheme, called Vertical and Pyramidal Summarization Tree (VP-ST). The first

step generates the summary of each cluster (or node), while the second step further combines the summary data into a more concise form. Each summary information is also represented in the form of a summary ChainNet. Given a query image, we first locate the partitions that contain images that are relevant to it. This is done by comparing its ChainNet with that of the summary ChainNet at internal nodes. Finally, the relevant partitions are examined.

We implemented a prototype WWW image retrieval system, called ICICLE (Image ChainNet & Incremental Clustering Engine) that employed the proposed mechanisms. We evaluated the system on a collection of 10,000 images obtained from documents identified by more than 2,000 URLs. Our results show that ICICLE is both effective and efficient compared to existing techniques. In particular, the Weight ChainNet model outperforms known techniques - Vector Space Model (VSM)[60] and [51] in terms of recall and precision. Moreover, the relevant feedback mechanisms can lead to significantly better retrieval effectiveness. In addition, ICC can also lead to faster retrieval time without sacrificing on the quality of the images retrieved.

The rest of this chapter is organized as follows. In Sections 3.2, we present our models for retrieval effectiveness, including image semantic representation model, the similarity measure, and the relevance feedback approaches to refine queries for further retrieval. In section 3.3, we present our efficiency methodologies, including incremental clustering algorithm and the summarization technique. In Section 3.4, we present the prototype system - ICICLE. In Section 3.5, we report the results of a performance study conducted on ICICLE. ICICLE is futher extended to take visual features into consideration in 3.6, and finally, we summarize this chapter in Section 3.8.

Symbols	Description
LC	Lexical Chain
TLC	Title Lexical Chain
ALC	Alt Lexical Chain
PLC	Page title Lexical Chain
SLC	Sentence Lexical Chain
RSLC	Reconstructed Sentence Lexical Chain
QLC	Query Lexical Chain
N	Number of images
L	Length of Lexical Chain
D	Number of Lexical Chain in an image
C	Number of Cluster
U	Upper bound of cluster size

Table 3.1: A Table of Notations in Chapter 3

3.2 ChainNet: A Semantic Model for WWW Images

Two key issues must be addressed in designing an effective image retrieval system to support WWW images:

- Determine a representation for a WWW image and the query semantics.
- Determine a similarity measure between an image and a query based on their representations.

To further improve the precision, Relevance Feedback (RF) is an important tool.

Before we move to next, here we provide a table (as shown in Table 3.1) which contains the used notations for easy reference.

3.2.1 Image Representation Model

An effective semantic model for WWW images must possess several desirable properties:

- **Exactness**: To be effective, it has to capture the essential image/query semantic meanings.

- **Computationally inexpensive similarity matching**: It should be fast to compute the similarity between the representations.
- **Preservation of the similarity**: The similarity between the image/query semantic meanings should be preserved for the representation.
- **Automatic extraction**: The representation should be automatically extracted, rather than manually generated.
- **Insensitivity to noise, distortion, rotation**: Any noise or distortion should not affect the representation drastically.

To understand the relationship between an image embedded in a HTML document and its surrounding text, we conducted a preliminary study on a collection of images obtained from HTML documents, which are from domains of BBC news and World Travel Guide. We randomly choose 500 images and to see if the embedded image is well described by its surrounding text. Based on our findings, out of 500 images, 94% can be semantically identified by the combination of the following four textual parts for our image collection. These are:

- **Image title**: Image file title (simply image title) is a single word that basically indicates the main object that the image is concerned with.
- **Image ALT (alternate text)**: The image ALT tag in HTML document is a phrase that usually represents an abstract of the image semantics.
- **Image caption**: The image caption usually provides the most semantics about an image. It is the image's surrounding text in the HTML document. It can range from one sentence to a paragraph of text that contains many sentences.

- **Page title**: Since images are used for enhancing the Web page's content, page title is most probably related to the image's semantics. It is usually a short sentence that summarizes the Web page's content.

From our founding, we can see that at least 6% of images in WWW may be missed in text-based retrieval because the text is not informative. Content-based searching can be used to reduce this ratio. However, this is not this article's focus. We argue that these four parts are reliable to represent image's semantic since 94% images can be well described by them. There are also some other parts which may provide some information about the image, such as other HTML meta data, However, they contain too much unrelated information. We have also excluded the textual content of the whole HTML document as part of the image's semantics for the same reason, i.e., that some information may be completely unrelated to the image content, and indexing the whole HTML document for each image in a very large database is not expected to provide an efficient solution. Therefore, we just use these four parts to represent image content. We note that all these four parts – image title, image ALT, page title and image caption – can be automatically extracted from the HTML document based on hypertext structures.

Based on the above findings, we propose representing the image semantics using the Weight ChainNet model that is based on the concept of lexical chain [36, 57]. Figure 3.1 illustrates an example. [36] defined a Lexical Chain (LC) as a sequence of semantically related words in a text. Inspired from that, here we re-define a LC as a sentence in image's surrounding text. In grammar, a sentence can be regarded as a sequence of semantically related words. As an image title is just a single word, we say it's a trivial lexical chain - Title Lexical Chain (TLC). The text obtained from the ALT tag is referred to as the Alt Lexical Chain (ALC). The page title is represented as a LC too - Page Lexical Chain (PLC). Finally, since a caption comprises multiple

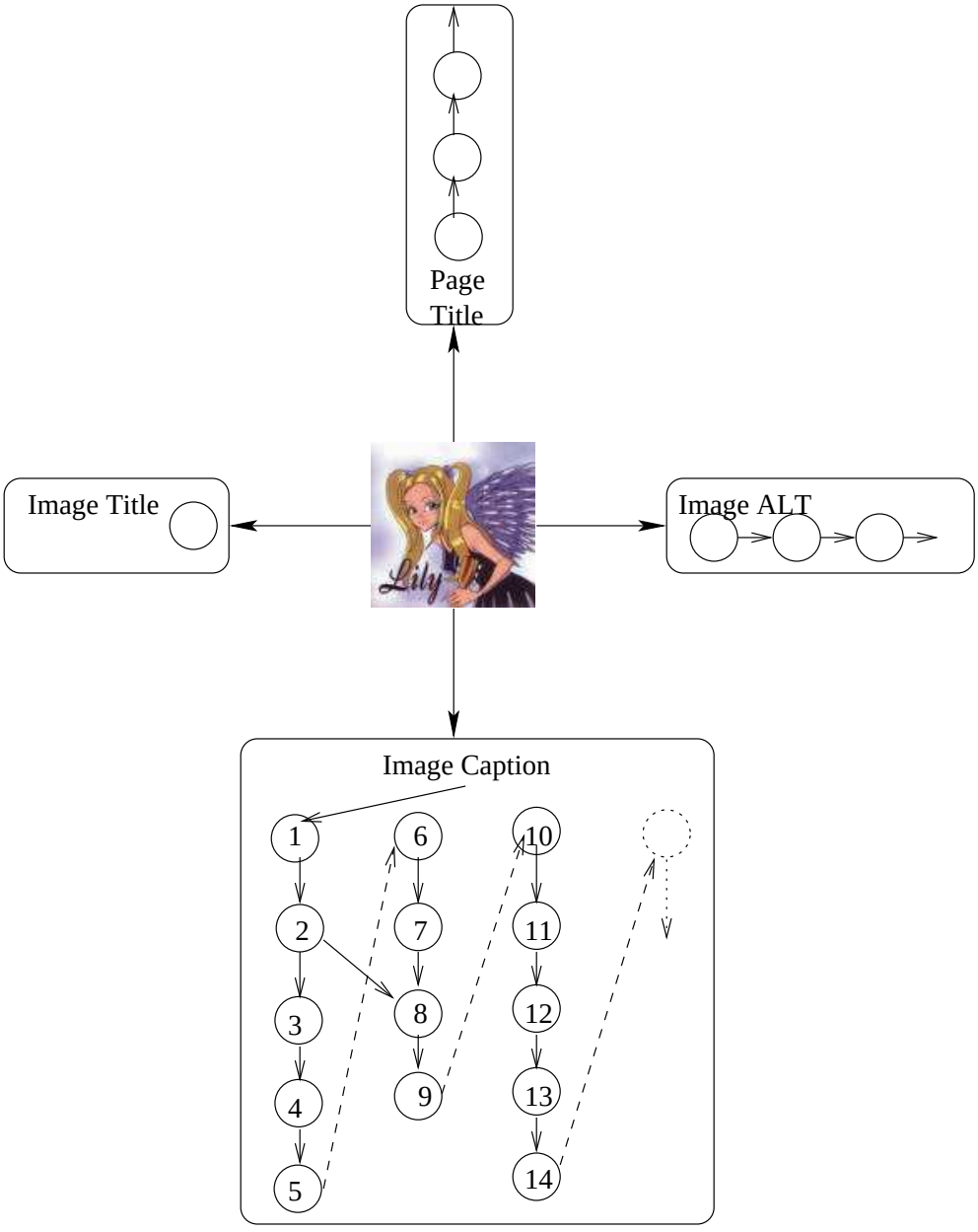


Figure 3.1: Image Semantic Representation Model - Weight ChainNet

sentences, we represent it as three types of lexical chains. Type one is called Sentence Lexical Chain (SLC), which represents one single sentence in an image caption. In Figure 3.1, each sentence is shown as one column in the caption component, i.e., each column is a SLC. Type two is called Reconstructed Sentence Lexical Chain (RSLC), and it represents one new sentence reconstructed from related sentences. RSLC was inspired from that two sentences are usually related if both share one or more words. One identical word in two SLCs splits each SLC into two. Based on the first common word, the second SLC's second half is connected to the first SLC's first half to form a RSLC (Notice that the SCLs are checked in order). In Figure 3.1, a RSLC exists if there is an arrow from one column to another column. The last type is called caption lexical chain (CLC), which represents the whole image caption. A CLC is formed by connecting SLC one after another. In Figure 3.1, the connections are made by dotted arrows. To illustrate, the followings are some examples from Figure 3.1: SLC (1→2→3→4→5), RSLC (1→2→8→9), CLC (1→2$\cdots$13→14).

The ChainNet model is built by these 6 types of lexical chains. Each chain captures a portion of the semantic structure of the image. A TLC indicates the main subject of an image. An ALC provides short description about an image. A PLC shows part of its content. An SLC captures the semantics of a single sentence in the image caption. An RSLC captures related sentences' semantics, and a CLC keeps the image's overall semantics. That's why we call it ChainNet, which is basically made of a chain of LCs. A real example can be found in Figure 3.2 and corresponding Table 3.2.

However, the ChainNet treats each type of LC as of equal importance, Now, simply representing an image in this way without capturing the relative importance of the various components is not expected to lead to good performance. For example, the image title, ALT, page title and image caption play different roles in representing an image's semantics. The reason we have divided the entire image caption into three

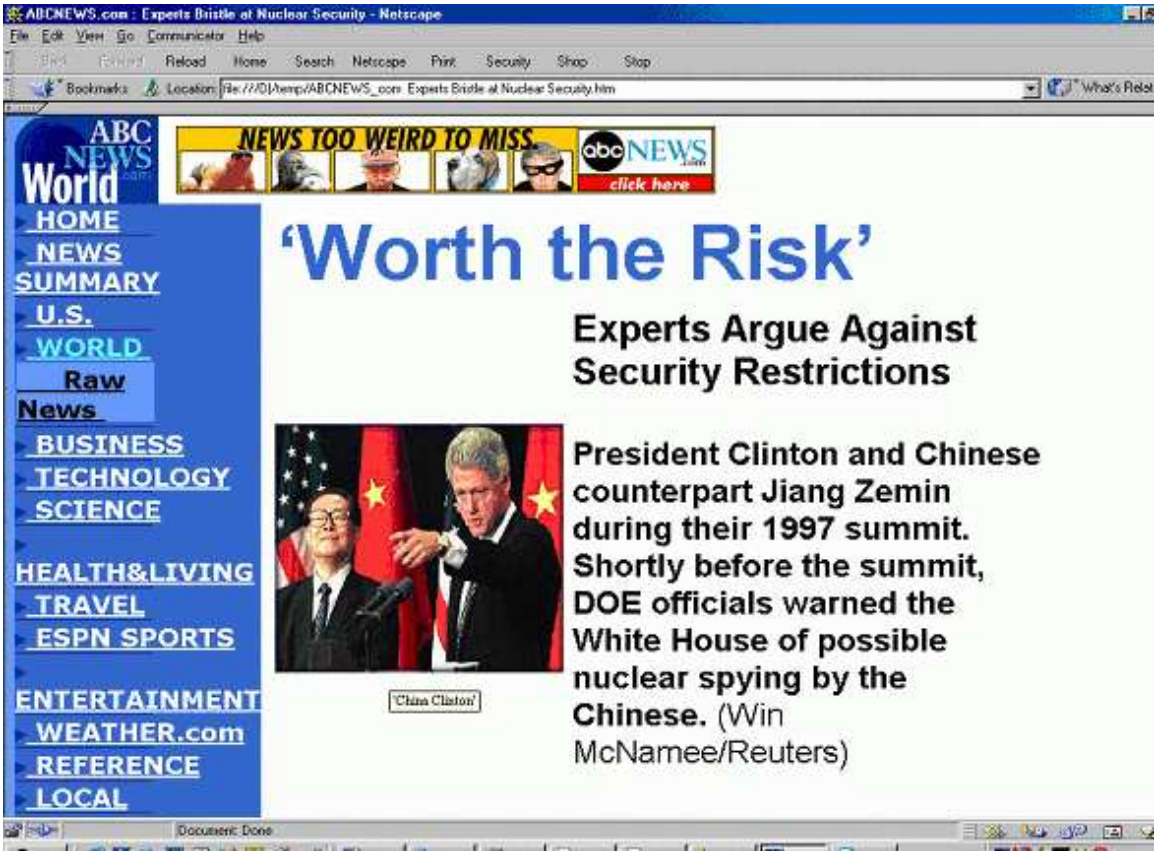


Figure 3.2: An example WWW image from ABCNEWS Website

LC Type	Description
TLC	ap_china_nuclear_h
ALC	China Clinton
PLC	ABCNEWS Experts Bristle at Nuclear Security
SLC	President Clinton and Chinese counterpart Jiang Zemin during their 1997 summit
	Shortly before the summit, DOE officials warned the White House of possible nuclear spying by the Chinese
RSLC	None
CLC	President Clinton and Chinese counterpart Jiang Zemin during their 1997 summit. Shortly before the summit, DOE officials warned the White House of possible nuclear spying by the Chinese.

Table 3.2: LCs in ChainNet of the image in Figure 3.2.

types of lexical chains is that we want to differentiate the importance of each type of sentences due to their positions and inner relationship within an image caption. The three types of lexical chains in an image caption are not equally important. The importance order from high to low is expected to be like this: SLC > RSLC > CLC. For instance, if all the same words in a query appear in an SLC, an RSLC and a CLC respectively, the SLC possesses the most semantic meanings among the three, followed by RSLC and finally CLC. For example, if a query is contained in an SLC (CLC also) in the first image, is contained in an RSLC (CLC also, but not SLC) in the second image, and is contained in a CLC (but not RSLC or SLC) in the third image, it's most likely the case that the first image is most relevant to the query, followed by the second image and then the third image, because an SLC is more semantically structured than an RSLC, which is more semantically structured than a CLC.

To capture the relative importance of the various types of LCs, we assign weights to the various LCs such that LCs that are deem to be more representative of the image content are assigned larger weight values. We shall see how these weights come into play in the similarity measure to be discussed in the next section. We note that for the caption, one word in the caption may have up to three different weights with respect to the lexical chains it belongs to. Of course, each word has at least two weights: a SLC weight and a CLC weight. If the word belongs to one RSLC, it will have three weights. One image caption may have several SLCs and several RSLCs, but only one CLC. If only one SLC exists, CLC and SLC are identical. The different weight for each type of LC can be confirmed from figure 3.11 in section 3.5.

Thus, the resultant Weight ChainNet model can be formally defined as: a network that is composed of the collection of the different types of lexical chains with different weights. It uses a well-structured notion of image's content to capture the semantic relationship between an image and its nearby text. Such a model can be seen as a

semantic representation of the content of an image. This model has the properties of exactness, since it captures an image's essential semantic meanings by an image title, ALT, page title and caption. The content can be automatically extracted. Finally, similarity matching is computationally inexpensive using the proposed list space model that we shall introduce in the next section. For a user query, it's usually a free sentence that describes the image content. Naturally, we represent it as a Query Lexical Chain - QLC.

3.2.2 Semantic Measure Model

In this subsection, we will present our similarity measure model between two lexical chains, and between an image and a query respectively.

Similarity between two Lexical Chains

We have presented the model for representing image/query semantics. To calculate the semantic similarity between a query and an image, we start from determining the similarity between two basic components in an image ChainNet - LC. In our implementation, we store terms of each LC as a list. All the lists belonging to an image are connected to the image root as shown by the ChainNet model (see Figure 3.1). We propose a **list space formula** to compute the similarity between two LCs as follows:

$$Similarity_{list_1,list_2} \equiv \frac{\sum_{i=0}^{Set.size()} e_1.weight * e_2.weight}{\sqrt{list_1.size()} * \sqrt{list_2.size()}} * MatchScale$$

where Set is the set of all word pairs (e_1, e_2) such that e_1 is from $list_1$ and e_2 is from $list_2$ and e_1 and e_2 are same word. We say two words are *matched* if both are same. The e_1.weight is the weight of LC that e_1 belongs to. The denominator of the formula keeps longer lists from automatically getting higher similarities.

In the formula, one important parameter is considered: *MatchScale*. Match scale is defined as the closeness of two lists from the view of match order. For example, one LC is *"US president Clinton and wife visited China in 1997"*, and the other one is: *"China president Jiang Zemin welcomed Clinton and wife in Tian'an square"*. For these two LCs, there are four matching words. For the first LC, the matched words are in order of *"president Clinton wife China"*, and in the other, they are *"china president Clinton wife"*. We treat each one as a child LC of its original LC. Let v1 and v2 represent the child LC of the first and second original LCs respectively. That is, v1 is (president→Clinton→wife→China) and v2 is (China→president→Clinton→wife). Obviously, both child LCs have exactly the same set of words and the same size. However, the orders of matched words in the two child LCs are not the same. Obviously, the closer the matched order of two children LCs are, the closer the semantics of the original two LCs are. Inspired from the formula for the *angle between two nonzero vectors* in 2d-space, we define the match scale as below:

$$MatchScale_{v1,v2} \equiv \frac{v1 \bullet v2}{||v1|| * ||v2||}$$

where v1 and v2 represent the child LC of the first and second original LCs respectively. The element value in v1 and v2 is the *position* of the word in their respective LC. But the *dot product* between two child LCs is redefined as the following:

$$v1 \bullet v2 \equiv \sum_{i=1}^{v1.size()} v1_i * v2_j$$

where $v2_j$ is the matched word in v2 for $v1_i$ in v1. As mentioned, two words are matched as long as they are the same. For instance, the positions of words *"president Clinton wife China"* in v1 and v2 are (1,2,3,4) and (2,3,4,1) respectively. The *dot product* between v1 and v2 is 1*2+2*3+3*4+4*1=24. Since ‖v1‖*‖v2‖ is $1^2 + 2^2 + 3^2 + 4^2$=30, the MatchScale between v1 and v2 is 24/30=0.8. We can see

that 20% of the similarity is deducted due to different orders of words. Obviously, when two child LCs are same, MatchScale becomes 1.

The above measure determines the similarity between two LCs. However, the two LCs may not be semantically related. For example, consider the query "Singapore Map". An image about Singapore Food, say I1, that contains several occurrences of "Singapore" in CLC may result in a high similarity value even though the images are not semantically related. On the other hand, another image about Singapore Map, say I2, contains only one occurrence of "Singapore" in CLC may result in a lower similarity value despite the fact that it is a desired image. To ensure that two LCs are *semantically related*, we need another parameter called: **Match Level**. Match Level is the number of the distinct matched words by a LC and a QLC, denoted as: **LCMatchLevel (LC, QLC)**. The *match level threshold* is the minimum match level for a LC to keep its original semantics. We say one LC is semantically related to a QLC, if and only if the LC's match level is equal to or greater than QLC's match level threshold. Therefore, in our semantic measure model, semantic similarity for a LC with respect to a QLC is indicated by the similarity calculated by list space formula in its match level. The match level determines if the LC is semantically related to the QLC. And the similarity calculated by list space model shows how well it is semantically related to the QLC.

Similarity between ChainNet and QLC

Now it is time to calculate the semantic similarity between an image and a query. From the discussion above, we know that an image is represented by a Weight ChainNet, and a query is in the form of a lexical chain. To calculate their similarity, we

use the following formula:

$$ImageSimilarity_{image,query} \equiv S(TLC, QLC) + S(ALC, QLC) + \sum_{i=1}^{SLC.number} S(SLC, QLC) + \sum_{i=1}^{RSLC} S(RSLC, QLC) + S(CLC, QLC)$$

where S is the similarity between two LCs. The image match level is defined as:

$$ImageMatchLevel(ChainNet, QLC) = MAX$$
$$(TLC.weight * LCMatchLevel(TLC, QLC),$$
$$ALC.weight * LCMatchLevel(ALC, QLC),$$
$$PLC.weight * LCMatchLevel(PLC, QLC),$$
$$SLC.weight * LCMatchLevel(SLC, QLC),$$
$$RSLC.weight * LCMatchLevel(RSLC, QLC),$$
$$CLC.weight * LCMatchLevel(CLC, QLC))$$

We say one image is semantically related to a query if and only if its match level is equal to or greater than the query's match level threshold. It has the similarity calculated by the above formula with the query in its match level. From the formula, we can see that a query is more related to the image with more related LCs.

3.2.3 Relevance Feedback

Because of the large image collection and the impreciseness of a query, it is important to provide mechanisms to help users in specifying their queries more accurately. One such mechanism is to exploit feedback from users based on resultant images returned from the initial query. By allowing users to indicate the relevant (and irrelevant) images, the original query can be refined to further improve the retrieval effectiveness. For this purpose, we develop two techniques: *semantic accumulation* and *semantic integration and differentiation*.

Semantic Accumulation

The first method, called *semantic accumulation*, allows the user to pick the most relevant image (from the user's subjective judgment) from the result of previous

retrieval as the feedback image. The method accumulates all the previous feedback images' semantics to construct a new query for the next retrieval. The resultant query is represented as a kind of ChainNet called Weight F/Q ChainNet (Feedback/Query ChainNet) since it is constructed by query and the feedback image's ChainNet. This kind of new query is represented in Figure 3.3.

Obviously, the combination of every entire ChainNet from each previous feedback images is tedious if the user searches again and again. More seriously, more noise will be added into the new query. Therefore, rather than a whole image ChainNet, we use just one single lexical chain which is most semantically related to the original query in the previous feedback image's ChainNet. This is calculated by the list space model. The steps for this method are:

1. Perform search using the F/Q Weight ChainNet (or Weight ChainNet for first attempt).

2. User selects the current feedback image.

3. Construct the feedback image's Weight ChainNet

4. Extract the closest lexical chain to the original query from the feedback image by list space model.

5. Use the QLC and the feedback image's weight ChainNet to construct F/Q ChainNet

6. Use that extracted LC in step 4 and old QLC to construct new QLC by adding together from head to tail

7. Go to step 1

In this algorithm, the semantic is accumulated by adding one most related LC from every previous ChainNet to QLC to form a new QLC. Therefore, the QLC carries richer and richer semantics as users provide more feedback.

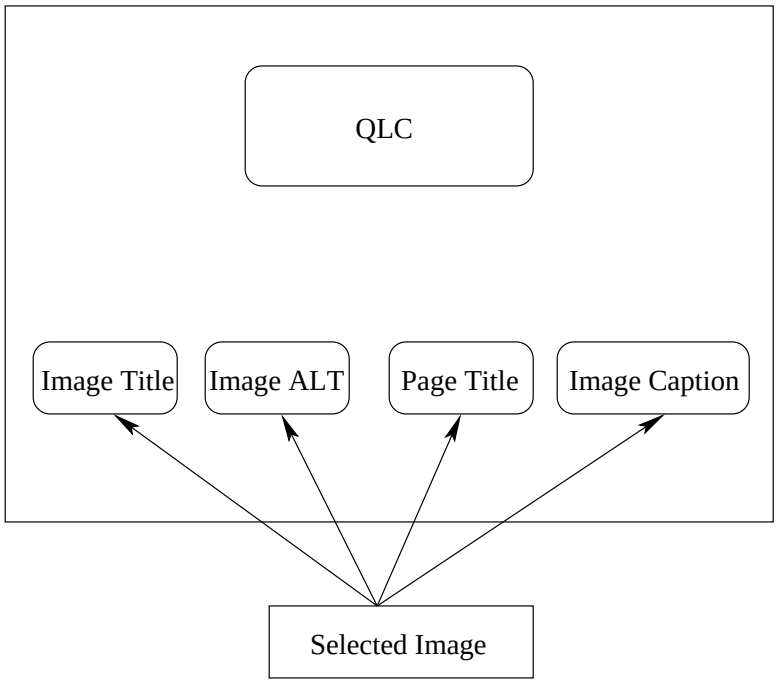


Figure 3.3: F/Q ChainNet in Semantic Accumulation

Semantic Integration and Differentiation

In the semantic accumulation feedback approach, users can only select one image at a time as the feedback information. To save time and to filter more unrelated images, we introduce another technique: semantic integration and differentiation. In this method, users can select several relevant and irrelevant images simultaneously. By relevant, we mean images that are semantically related to the query as judged by the user and hence should be retrieved. On the other hand irrelevant images are those that the user considers to be unrelated and should not have been retrieved. The system integrates the related semantics obtained from the relevant feedback images to construct a new query for the next try. After that, the system combines the semantics

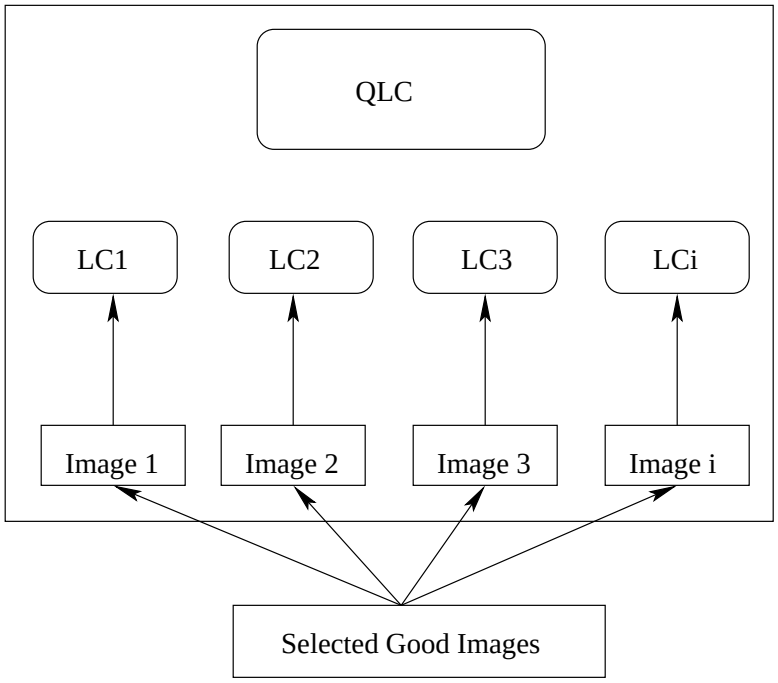


Figure 3.4: F/Q ChainNet in Semantic Integration and Differentitaion

from irrelevant images to differentiate the irrelevant images from the returned results. The new query is also represented by a F/Q Weight ChainNet as shown in Figure 3.4.

The steps for this method are:

1. User selects a number of relevant and irrelevant images.

2. Extract the most semantically related LC from each relevant image's ChainNet to form a new F/Q ChainNet with QLC as a new query.

3. Extract the most un-semantically related LC from each irrelevant image's ChainNet to form a ChainNet for bad images.

4. Submit the query

5. From each returned image, remove it from results if it's more related to the bad images' ChainNet.

6. Go to step 1

The semantic similarity formula between two ChainNets can be easily extended from the formula for measuring the similarity between an image and a query.

3.3 ICC: Incremental Clustering of ChainNet

In this section, we present our approach to speed up the retrieval process. As current indexing methods, like inverted file, are all based on analysis of words (mostly nouns and verbs), they are not applicable to ChainNet, where the basic unit is LC and the semantic relationship between LCs should be kept. So as to the current incremental clustering techniques. Here our basic idea is to partition the image collection so that only the relevant partitions need to be examined during the retrieval process. Instead of predetermining the partitions (e.g., fixing the number of partitions and the types of images that belong to the partitions), we propose using a clustering strategy based on ChainNet. Moreover, our clustering strategy can dynamically adapts the number of partitions and partition sizes. We shall use the example shown in Figure 3.2 as our running example in this chapter. Its corresponding Weight ChainNet representation is shown in Table 3.2.

3.3.1 Incremental Clustering Algorithm

We define the incremental clustering problem as follows: for a continuously growing data sets, maintain a collection of reasonable number of clusters and cluster size corresponding to data size. Whenever a new point is presented, either it is assigned to one of the existing clusters or treated as noise, or it triggers a merge or split process. Our Incremental Clustering on ChainNet (ICC) model has the following salient features: (1) it can construct arbitrary shaped cluster, (2) it handles the temporary "noise" by using a 'noise box' we introduced, (3) its overall cluster structure is indexed in

a Hierarchical ChainNet Summarization Tree (HC-ST), (4) it can adapt the number of clusters and the cluster sizes dynamically, (5) it is not sensitive to the data order, and (6) it requires little main memory storage and has O(N^2) time complexity.

Figure 3.5 shows the main flow of the ICC algorithm. ICC starts as the first data point comes into the database. Initially for the first few points, we treat them as noise. There are several issues involved in this routine. (1) how to identify a cluster? (2) how to decide when to merge and when to split? (3) how to merge and how to split? (4) how to decide when to construct a new cluster from the noise dataset? (5) how to update the HC-ST in all cases? In the following, we will discuss them one by one.

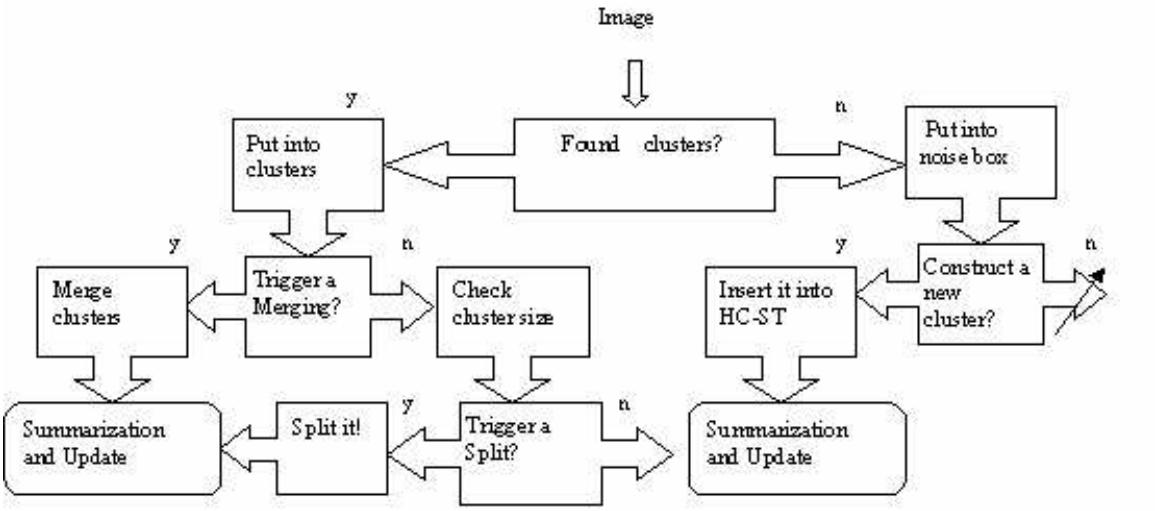


Figure 3.5: ICC Main Routine.

Hierarchical ChainNet Summarization Tree Structure

In ICC, we employ a hierarchical tree structure, Hierarchical ChainNet Summarization Tree, to facilitate the clustering and retrieval process. Figure 3.6 shows a HC-ST

structure. In HC-ST, the leaf nodes are the summary information of clusters, one leaf node per cluster and each leaf node is linked to its corresponding real data level. The higher level (or internal) nodes are the summarized representation of all its children nodes. We shall defer the discussion on how the summary information is obtained to the next subsection.

All the nodes employ the same uniform data structure - ChainNet. The root node will represent the whole database's description. As we may notice, there is a 'noise box' in HC-ST. This 'noise box' contains all the temporary noises that are not related to any of the current clusters. Completely new clusters (i.e., those not related to existing clusters) are generated from the 'noise box'. This HC-ST starts with the generation of new clusters from the 'noise box', and is updated by merge and split operations. It may not be balanced as the split process goes on, but it always keeps a *binary* tree (except for the root node). The Split operation is a new operation that never occurs in static clustering methods. This hierarchical tree structure can speed up the process of finding the right cluster. To construct this tree, there are four main operations: insert, merge, split and generate new cluster. The first thing, of course, is to find the right cluster given a new point.

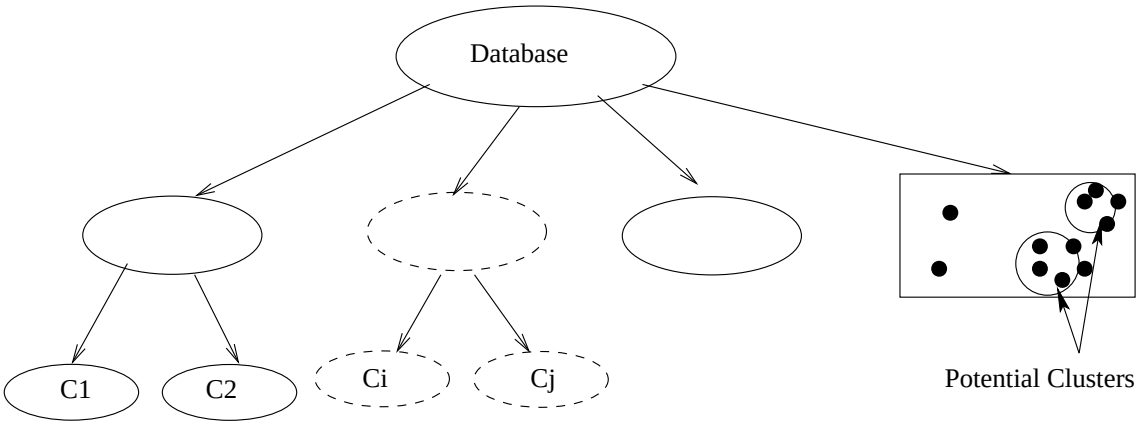


Figure 3.6: Overview of HC-ST.

Find the Right Cluster

When a new image is to be inserted, we first determine which cluster it belongs to. ICC will traverse the HC-ST from the root until either a relevant cluster is found or no relevant internal node is found. To decide whether an image's ChainNet is semantically related to any node's ChainNet, we use the proposed List Space Model with MatchScale. If the sum of the similarity and MatchScale is not less than the relevance threshold, the two ChainNets are semantically related. This threshold is a linear function of the query's ChainNet. The traversal will stop if an image's ChainNet is not related to any child of the current node. It's possible that an image's ChainNet is related to the current node, but not to any of its children, since the current node contain summarized information that is a combination of all its children.

Insert

If a right cluster is found, we insert it to that cluster and build interconnection with relevant images (i.e., all related images are linked); otherwise, we put it into the 'noise box' and check if there are any related noise or small group. If there exists related group, we build the interconnection too. Any connected group in the 'noise box' may potentially become a cluster.

Merge

When a new image matches two or more clusters, a merge phrase may occur. However, merging will only occur for clusters that are related (based on the list space model). If the clusters are not related, we assign the new point into the closest cluster. To perform the merging, we merge the one with a longer path into the other to avoid long path; the cluster in the longer path is then removed. A new cluster representative will then be constructed for the merged cluster. This is done by combining the two

old clusters' ChainNets to generate their summary information.(as shown in Figure 3.7, the cluster with longer path is merged with the cluster with shorter path and two clusters form a new cluster indicated as bold node). This will help to keep the tree as balanced as possible. Since the tree is binary (except root level), once the cluster has been removed from its original position, its sibling will move up to replace its parent (as shown in Figure 3.7, after its sibling moves up, the height of tree is reduced by one). Finally, all accessors of updated nodes are updated correspondingly.

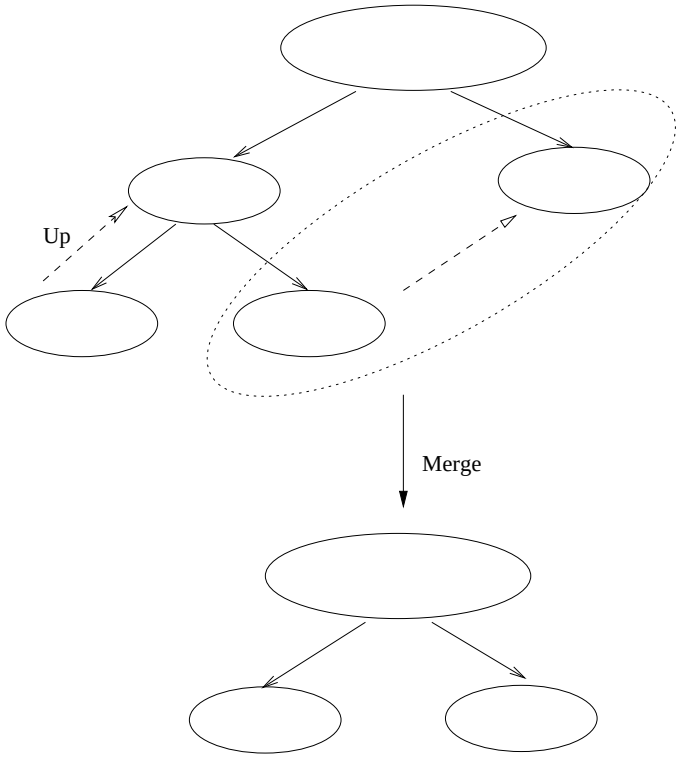


Figure 3.7: Illustration of the Merge operation

Split

For a large cluster, it is possible that it may comprise a number of smaller clusters. When a cluster's size keeps increasing and reaches a certain predetermined upper bound on the size of the cluster, the cluster may be split if it has smaller clusters. To decide whether a cluster is made up of smaller clusters, we need not go through

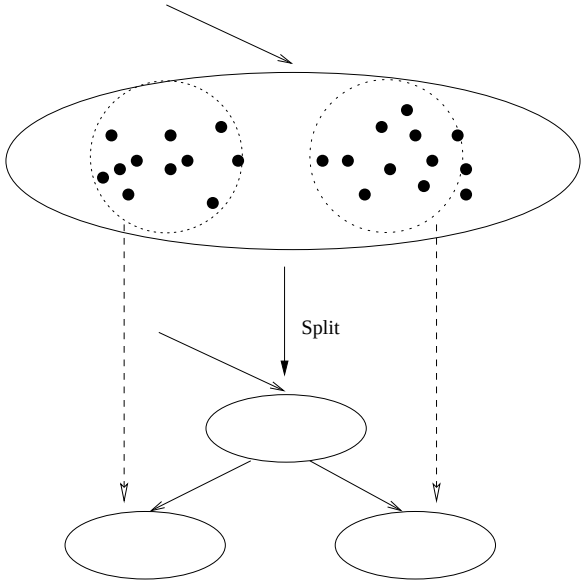


Figure 3.8: Illustration of the Split operation

the data sets at all. By analyzing the cluster's ChainNet structure, the Split process breaks a large cluster node into two smaller clusters, and replaces the original node with the parent node of the two newly created smaller clusters (see Figure 3.8). The Split process forces the cluster's data structure in a binary hierarchical tree.

We note that there is a correlation between the number of clusters (denoted as C) and the upper bound on the size of a cluster (denoted as U). As the database keeps growing, it is better for C and U to be correspondingly increased. If we fix the cluster number, its size will be too small and too large during the early and late stage respectively. On the other hand, there are too few and too many clusters for early and late stage respectively. We found that it's not so good to define U as the linear function of C. As more images are inserted into database, linear relationship will drive U to increase very slowly, which leads to rapid jump on C. Clearly, too many clusters will increase the height of the tree. Exponential function can express the relationship more by nicely controlling the height of the tree. Hence, we define U

as:

$$U = \alpha * C^2$$

where α is a scalable parameter. This formula provides a good adjustment between cluster number and cluster's upper bound. The relationship among data size (denoted as N), cluster number (denoted as C), and cluster upper bound (denoted as U) can be shown as:

$$N < U * C = \alpha * C^3$$

Generate new cluster

We define the upper bound to control the cluster's size and indicate whether it's time to split a large cluster. We also define the cluster lower bound to decide when to generate a new cluster from the 'noise box'. In the 'noise box', either all points are not related, or a few points are connected to form small groups, but the number is fewer than the lower bound. These small groups are potential clusters. While small group keeps growing and reaches the lower bound, a new cluster is generated and connected to the root node in HC-ST. Here we relax the binary tree to be a tree in which each internal node has two children, except for the root node.

Update HC-ST

For all the operations mentioned above, updating HC-ST is necessary, from the leaf nodes to the root node. For a new image assigned into a cluster, the system will generate a new summarized ChainNet for the cluster by combining the image ChainNet and the summarized ChainNet of the cluster (see next subsection). All internal nodes' summarized information are updated in the same manner.

For merging, all the original cluster's ancestor nodes have to extract its semantics and all the newly created cluster's ancestor nodes have to add in that cluster's

semantics. For splitting, it is simpler. The only thing to do is to create two new sub-clusters' representatives, and treat the original cluster's ChainNet as their parent. There is no need to update the ancestor nodes.

3.3.2 Summarization of ChainNet

To construct Hierarchical ChainNet Summarization Tree Structure (HC-ST), summarization technique has to be proposed. Here, we present a two-step mechanism, called Vertical and Pyramidal Summarization Tree (VP-ST), that is used to generate summary data for the nodes of HC-ST. The goal here is to create compact and short, but effective representation. The difficulty is how to retain as much semantics as possible. VP-ST will summarize all the data points within one cluster and produce the cluster's summary information - cluster representative, as the leaf nodes of HC-ST. Leaf nodes are then further summarized to form upper level nodes, and so on so for, until the root is reached. The proposed VP-ST mainly consists of two steps: Vertical summarization, followed by Pyramidal summarization (see Figure 3.9).

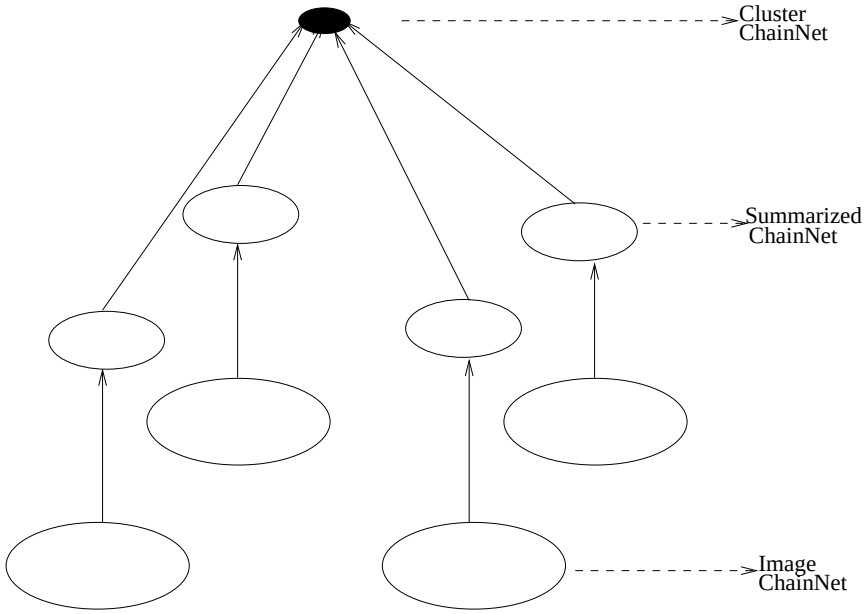


Figure 3.9: VP-ST structure

Vertical Summarization

In this step, we will summarize each image's ChainNet into a summarized ChainNet. As shown in Figure 3.9, the original ChainNet is the node at the lowest level, and the summarized ChainNet is the middle-level node. Recall that the ChainNet comprises 6 types of LCs. Our solution is to reduce the LC length, and the number of LCs. To make a successful summarization, we will stress on these two means one after one. We shall illustrate how the LC length can be reduced using the example in Figure 3.2, and the initial set of LCs of the ChainNet in Table 3.2.

Recall that an LC is basically a sentence that carries certain semantics by its words. Since RSCL and CLC are generated from SLCs, the key is to reduce SLCs. It's known that in a sentence, the most important parts are subject, verb, and object. All other information is auxiliary. Therefore, we parse each LC into several parts: subject, verb, object and some auxiliary parts. Auxiliary parts can be identified by prepositions in LC. Auxiliary parts will be removed from the LC provided they are not semantically related to any other LC without considering CLC and RSLC. For instance, the second SLC can be parsed as 6 parts. The subject is "DOE officials", the verb is "warned", and the object is "the White House", and there are 3 auxiliary parts: "Shortly before the summit", "of possible nuclear spying", and "by the Chinese". After computing the similarity by using the list space model, "of possible nuclear spying" and "by the Chinese" can probably be retained. This will depend on the threshold value that the user set. The last one "Shortly before the summit", should be removed from that SLC. The same process is applied to all the other LCs. Finally, the new ChainNet LCs is shown in Table 3.3 after reducing the LC length, stop words, and applying word stem (we assume "by the Chinese" is removed. "Nuclear" word appears both in ALC and PLC).

The next step is to reduce the number of LCs. There are two ways to do so.

LC Type	Description
TLC	ap_china_nuclear_h
ALC	China Clinton
PLC	Experts Bristle at Nuclear Security
SLC	President Clinton Chinese counterpart Jiang Zemin
	DOE official warn White House nuclear spy
RSLC	None
CLC	President Clinton Chinese counterpart Jiang Zemin. DOE official warn White House nuclear spying.

Table 3.3: LCs after Vertical Summarization step for Table 3.2.

The first is remove some LCs. It was shown in [50] that TLC, ALC and the first SLC play the most important roles in identifying an image's semantics. They cannot be removed directly. If one LC is not semantically related to any other LC (except CLC), it should be removed. In this example, the second SLC has little similarity with TLC and PLC. It is likely to be removed. PLC is similarly dealt with. The other approach is to combine similar LCs. If one LC's semantics can be identified by another similar LC, then we can merge one into the other by assigning the common words' weight as the sum of both LCs' weights, and move uncommon words into the other also. In this example, ALC in fact is contained completely in the first SLC. Therefore, there is no need to keep ALC. But we have to increase the common words' weights by adding ALC's weight. If there is only one SLC left, keeping a single copy is enough. Therefore, CLC is not needed any more. And the weight for SLC becomes the sum of both SLC and CLC. Finally, the summarized ChainNet for the example image is shown in Table 3.4 (we assume the second SLC is removed).

Comparing to the original ChainNet, some information is lost, like "white house" and "DOE official". But by the inter-relevance, we can confirm that that information is not relatively important in determining the image semantic. By adjusting relevance threshold, we also can decide how much we want to lose. This will definitely affect the retrieval precision. The point is how much this will affect. We will show such effect

LC Type	Description
TLC	ap_china_nuclear_h
PLC	Experts Bristle at Nuclear Security
SLC	President Clinton Chinese counterpart Jiang Zemin

Table 3.4: The final summarized ChainNet for image in Figure 3.2.

in the performance study. The purpose of generating each summarized ChainNet for each image is to try to represent all the points as close as possible.

Pyramidal Summarization

After we have obtained all the summarized weight ChainNet for each data point in the cluster, it's time to integrate them into one. In this phase, we want to keep as much information as possible. Every LC is very short, and important in deciding its image's semantics. Since all images within a cluster are related, many LCs from different images must be related to each other. This provides much space for merging LCs. We apply the same methods as combining two LCs in the Pyramidal Summarization. In this step, no information is lost since all LCs are reconstructed into a more compact form.

3.3.3 Time and Space Complexity

The time complexity for list space model is O(L^2), where L is the length of LCs. To compute the similarity between a query and an image, we need O($L^2 * D$), where D is the number of LCs in the image ChainNet. Here L and D are very small compared to N, the data size. Since the HC-ST is binary in nature (except the root), given the number of cluster C, the average height of tree is $a * ln(C)$ where a is a linear factor. To find the right cluster, we need to traverse O($ln(C)$) nodes at most. Therefore, the time to find a cluster is: O($L^2 * D * ln(C)$) in average. Next is the update time. We take insert, which is the most frequent operation, as our example. Summarizing time

is the same as comparing two ChainNets, which is O($L^2 * D^2$). And we need to update $ln(C)$ internal nodes. Therefore, updating needs O($L^2 * D^2 * ln(C)$) too. So the overall time complexity is: O($L^2 * D^2 * ln(C)$) in average. For N data points insertion, we need average time complexity O($N * L^2 * D^2 * ln(C)$). In this time complexity, there is no square on the large parameter N. And we believe $L^2 * D^2 * ln(C)$ will be much smaller than N when database is very large. Therefore, we argue that our ICC is very efficient to construct its HC-ST.

3.4 Architecture of ICICLE

In this section, we present the architecture of ICICLE, a prototype system that we have implemented. Our overall system architecture is shown in Figure 3.10.

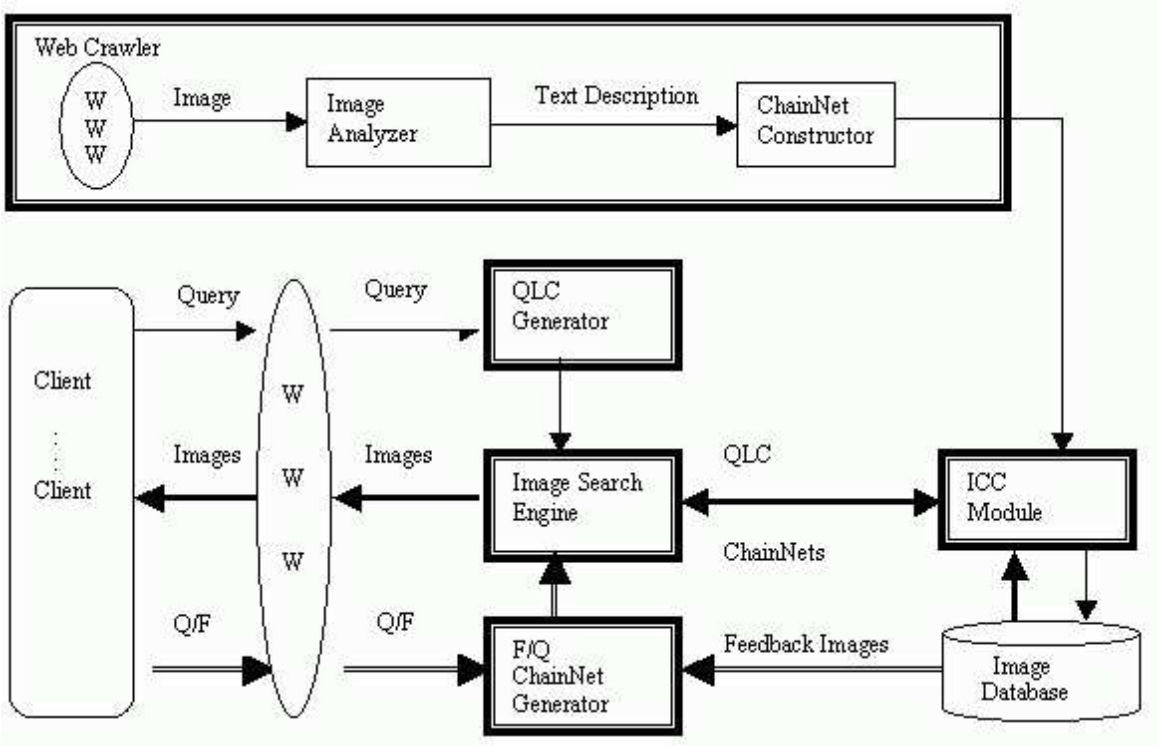


Figure 3.10: Overall ICICLE system structure in client-server form

In our implementation, the system consists of five basic components: a Web

Crawler, a QLC Generator, an Image Search Engine, a F/Q ChainNet Generator and an ICC Module. The database consists of three parts: WordNet, Query Profile, and Image Database.

The Web Crawler that operates in the background automatically searches the WWW for documents with embedded images. The crawler also extracts the image title, image ALT, page URL, page title and image caption from the HTML documents as the images' semantic content. After the above information has been extracted, image's ChainNet is constructed consequently. It then loads the meaningful images (or their URLs) with their representations into the database. For purpose of testing the model, we "centralized" the image collection (instead of simply extracting the image at runtime from the various Web sites/pages in the form of a search engine). In total, we collected 10,000 images from over 2,000 different URLs, which are mainly from web sites of BBC and World Travel Guide.

Whenever a new image comes, ICC module clusters it into the right partition. Whenever a query comes, the QLC Generator transforms a user query (free text description) to a query lexical chain (QLC). Image Search Engine then passes the QLC to ICC to allocate the right partition for comparison. The Image Search Engine compares the QLC against the ChainNet of the images returned from ICC module, and returns all semantically related images. The images are displayed in order of decreasing degree of similarity. Finally, the F/Q ChainNet Generator is used to generate extended query from the user query and the feedback images. Given the selected images from users, the F/Q ChainNet Generator creates a new weight ChainNet by combining the selected image's ChainNet, and the QLC. Depending on the feedback mechanisms adopted, different new weight ChainNet may be obtained. Finally, the Image Search Engine will perform the next round of retrieval based on the output of F/Q ChainNet Generator. A demo system has been shown in [49].

3.5 Performance Study

ICICLE employed the Weighted ChainNet model to capture the semantics of WWW images, and the Incremental Clustering on ChainNet model to speed up the retrieval process. We also study both the effectiveness and the efficiency of the proposed methods through an extensive performance study. We report our study and findings here. For the effectiveness improvement, we compared our image representation and measure models with [51]. We also compared our clustering strategy with two recently proposed clustering algorithms: CURE [24] and DBS [43].

3.5.1 Experimental Setup

We run the experiments on a 600 MHz CPU Pentium III PC with 128M RAM. In our experiments, we used 12 text descriptions, as shown in the following Table 3.5, as our queries. These queries are generated from the surrounding text of images which are randomly selected from database. Given that we have over 10,000 images, it is not practical to scan all images to obtain the relevant images for each query. To determine the set of relevant images for the queries, we adopt the following realistic approach. For each query, we expand the query terms to include terms that are related. This is done using the WordNet [35]. For example, the term girl may be expanded to include the term woman. Each term is then used as a query to extract the list of images whose semantics (or rather the LCs) contain that term. The union of the results from each term form a candidate set of relevant images. We then manually examine the candidate set to eliminate those that are not semantically related to the query to get the final set of relevant images. To remain the judgement fair, the examination of the candidate set was done by one person using human-judge, without looking at the text content of the images.

Query	Query Description
Q1	Singapore map
Q2	Travel in Spain
Q3	Valentine flower
Q4	Island in the sky
Q5	California beach girl
Q6	England football league
Q7	Green lizard on a red leaf
Q8	National University of Singapore
Q9	Hollywood superstar Jennifer Lopez
Q10	Husband is kissing his wife
Q11	Elephant in the beautiful national park
Q12	Celebrations for new millennium of 2000

Table 3.5: Test Queries.

3.5.2 Tuning the Weight ChainNet Model

In this set of experiments, we "turn off" the clustering strategy. This allows us to study the effectiveness of the Weighted ChainNet model in isolation.

Tuning the Weights

Weight ChainNet model calls for some tuning to be performed. As mentioned earlier, there are 6 types of LCs and different LC types may have different significance in identifying the image semantics. In the first experiment, we evaluate the performance of each type of LCs exclusively to study their different impact on retrieval effectiveness. Figure 3.11 shows the changes of precision and recall, as more images are returned for queries.

From Figure 3.11, we can see that for TLC, it cannot achieve >20% recall, although it has high precision. This is due to the lack of information in TLC. For similar reasons, PLC and ALC are not very effective also. For RSLC, since quite a number of images do not have RSLC, it cannot achieve high recall. Only CLC and SLC can result in high recall, but the precision is not satisfactory. TLC, ALC, PLC and RSLC can be used to improve the precision a lot. On the other hand, SLC and

CLC can improve the recall. From this result, we have a rough picture of the relative importance of each type of LCs. Clearly, SLC is the most important, followed by TLC, RSLC, ALC or PLC, and finally, it is CLC.

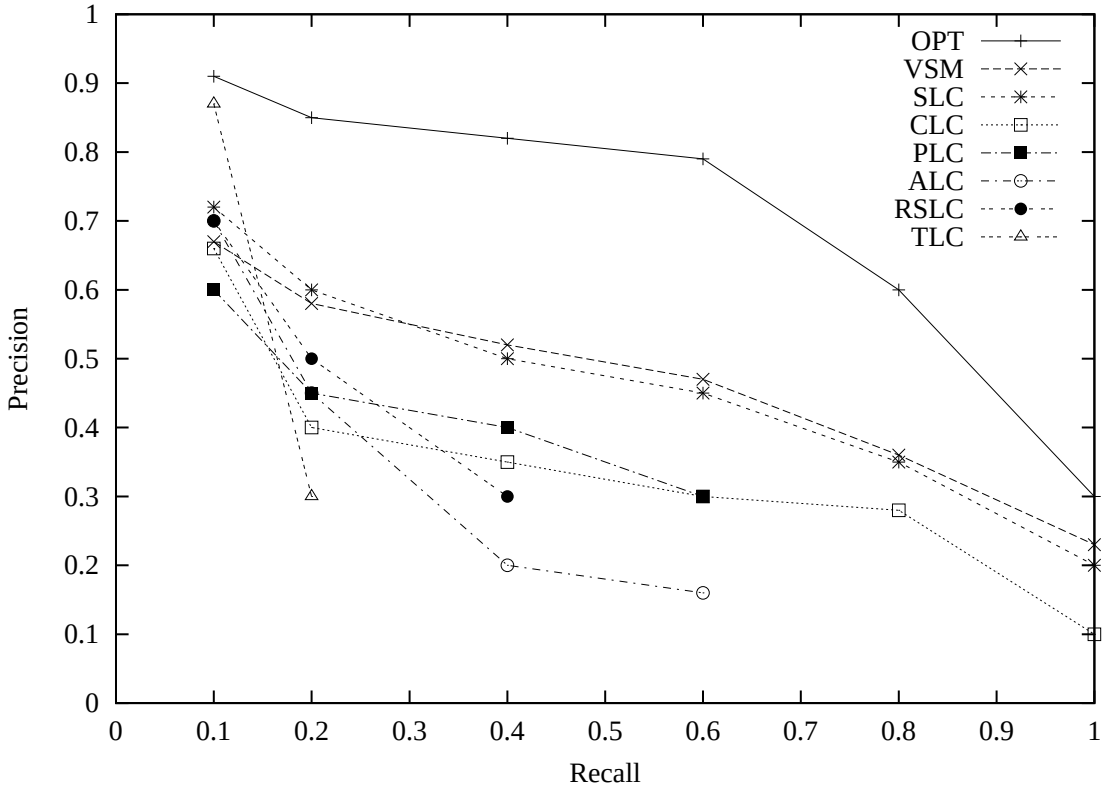


Figure 3.11: Utility by each Type LC alone to Represent Image

To determine the weights to be assigned to the proposed model that combines all the LCs, we tested different weight combinations from values in 0, 0.2, 0.4, 0.5, 0.6, 0.8 and 1 for each type of LCs. However, we narrowed the search space based on the result from Figure 3.11 by adopting some simple heuristics. For example, since SLC is the most important, we fixed its weight at 1.0. Moreover, for a LC that is more important, the weights assigned to the other less important LCs cannot be more than its weight. In total, we tested 22 combinations and obtained the following weight assignment for the various LCs: TLC (0.8), ALC(0.6), PLC(0.6), SLC(1), RSLC(0.5),

CLC(0.2). In this experiment, we have fixed the scale parameter coef of the match level to be 0.6 (see Tuning the Match Level). We shall refer to this scheme as OPT. We also presented the result of OPT in Figure 3.11. As shown, we can get more than 80% precision with recall of 60%. Clearly, the results show that using a single LC exclusively cannot provide the best performance, even though such an approach is clearly simple. Moreover, it shows that proper combinations of the various LCs can lead to very effective retrieval results.

Though this experiment is meant to tune the proposed method, we note that it is also a comparative study among the different schemes. We compare our OPT scheme with well-known text document representation model - Vector Space Model (VSM) [60]. However, VSM has to be extended so that it can be employed to represent an image. Using VSM, we constructs an image vector by considering all the words in the image's title, page title, image's Alt, and image's caption. The similarity between a query vector and an image vector is measured by using standard *cosine similarity* [60]. Figure 3.11 shows that the performance of VSM is close to that of SLC alone. Although VSM contains all the words of an image, its performance is unsatisfactory. The obvious telling reason is that VSM has completely lost the sematic structure among the words. However, the semantics are more carried by the relationships among words. We also note that the exclusive CLC scheme can be viewed as a form of text-based system [51] without any semantic structure involved, which is even much worse than VSM. This is because that VSM also includes the information of image title, image Alt and page title, except image caption. Obviously, OPT outperforms the existing schemes - VSM and [51]. This confirms that our WWW image representation model is more effective.

Tuning the Match Level

There is another parameter that we have to tune, the match level. Recall that the match level is the number of common terms shared by two lexical chains. It determines whether two LCs are semantically related, and then derives if two images are semantically related. In our evaluation system, only those semantically related images are returned.

One single word cannot reflect the semantic meaning of a whole query. If the match level threshold is too small, too many images may be returned to the users. On the contrary, too few images are displayed if the match level threshold is too high. Therefore, it is necessary to choose the best match level thresholds. Since the length of a query is a random variable, a fixed value for match level is not applicable to various queries. We thus define the match level as a linear function of query length:

$$MatchLevelThreshold = coef * query.length() + constant$$

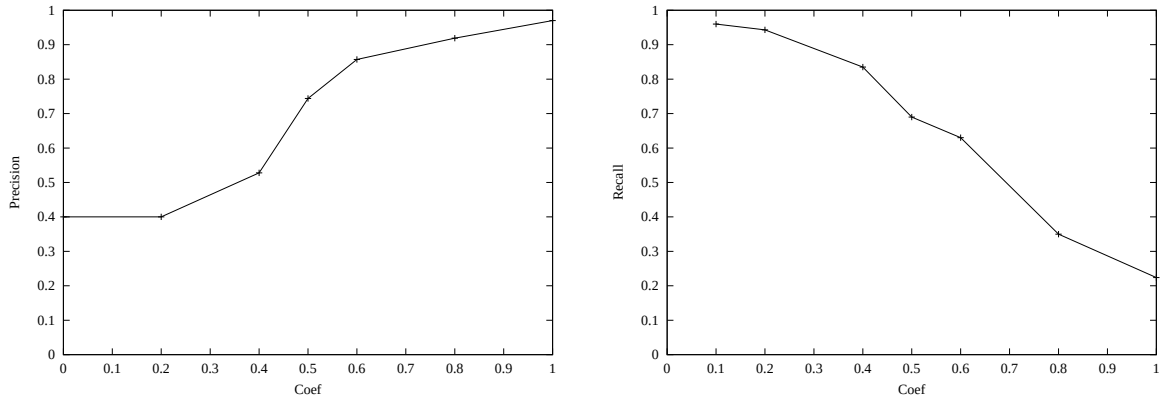
where the $coef$ is the scale parameter we need to explore in order to get the best results in a reasonable volume. And the constant is just an adjustable value.

We tested those 12 queries in Table 3.5 in order to select the best coef. Figures 3.12a and 3.12b shows the relationship between precision and $coef$, and recall and $coef$ respectively.

From Figure 3.12a, when coef is > 0.6, the precision will be greater than 85%. From Figure 3.12b, we can see that when coef is < 0.6, the recall is greater than 60% which is very satisfactory to a large image database. Therefore, observing the combined effect, we select 0.6 as the optimal value of coef.

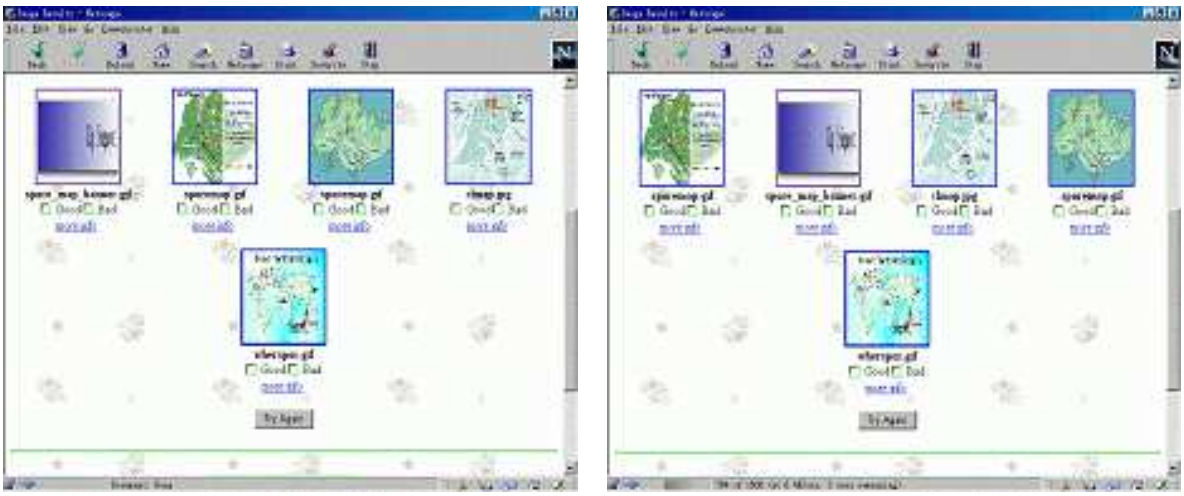
Impact of Match Scale

Match scale explores the importance of match order in the lexical chain. It has the effects in terms of image ranking during presentation of the returned images. Images



(a) Precision Vs. Coef (b) Recall Vs. Coef

Figure 3.12: Effect of Match Level.



(a) Q1 results before Applying Match Scale (b) Q1 Results after Applying Match Scale

Figure 3.13: Effect of Match Scale.

with higher similarity measures will be returned to users ahead of images with lower similarity values. Figure 3.13 shows a sample results obtained from Q1. As shown, by considering the match scale, we can get more relevant images being displayed earlier, i.e., ranked higher.

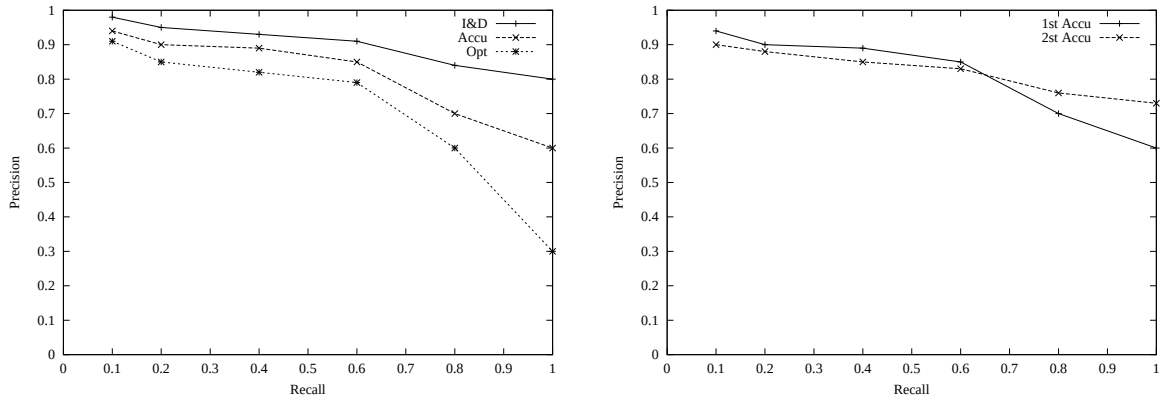
3.5.3 Feedback Mechanisms

In this experiment, we study the effectiveness of the two proposed feedback mechanisms: *semantic accumulation* and *semantic integration and differentiation*. Figure 3.14a shows the improvement by the two methods respectively. OPT is the basic Weight ChainNet model without feedback. *Accu* denotes the semantic accumulation method. And $I\&D$ represents the semantic integration and differentiation method. We note that *Accu* and $I\&D$ represents one application of the feedback loop after OPT returns its resultant images.

To clearly see the effect of the noise that semantic accumulation brought, Figure 3.14b presents the results for the first feedback and second feedback by semantic accumulation. We can see that the second searching actually has a bit lower precision, but with relatively higher recall. But semantic accumulation method has the advantage that the returned images are more semantically related to the specific image selected by user - the feedback image.

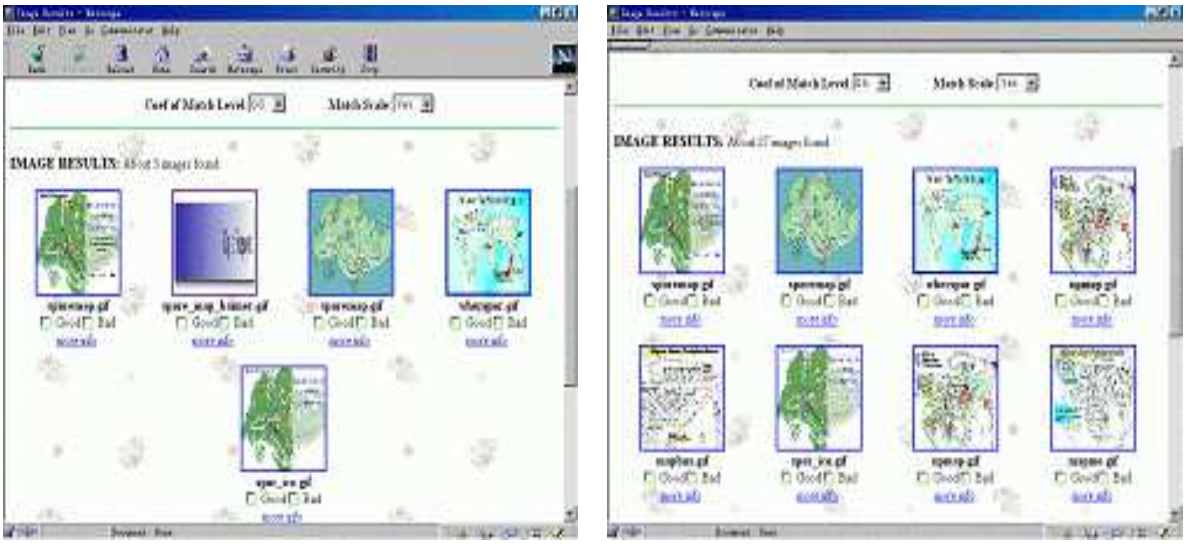
Figure 3.15a shows a sample feedback run of the *Accu* method for Q1. Compared to the results generated from OPT (the basic Weight ChainNet model without feedback, see Figure 3.13b), we see that the set of images retrieved are more relevant.

Figure 3.15b shows a sample feedback run of the $I\&D$ method for Q1. As we can see, the resultant images are not only more relevant than OPT and *Accu* approaches, the irrelevant image has also been pruned. In addition, more relevant images have been retrieved.



(a) Comparison of feedback mechanisms (b) 1st and 2nd try by Semantic Accumulation

Figure 3.14: Effect of Feedback Mechanisms



(a) One-step feedback of Accu for Q1. (b) One-step feedback of I&D for Q1.

Figure 3.15: One-step Feedback Results for Q1

3.5.4 Comparative Study on Clustering Techniques

In this experiment, we compare ICC with two static clustering algorithms proposed recently: CURE (Clustering by Representatives) [24] and DBS (Density Biased Sampling) [43]. For both CURE and DBS, sampling is used. We set the default sample size to 125, and the number of representative per cluster is 10, as indicated in [24]. We also compared against the original ChainNet model (i.e, without ICC).

Effect on Retrieval Effectiveness

We note that the retrieval effectiveness with and without ICC can be different. This is because, under ICC, only some relevant partitions will be searched. It is not impossible that some relevant images fall into a partition that is not examined during the retrieval process.

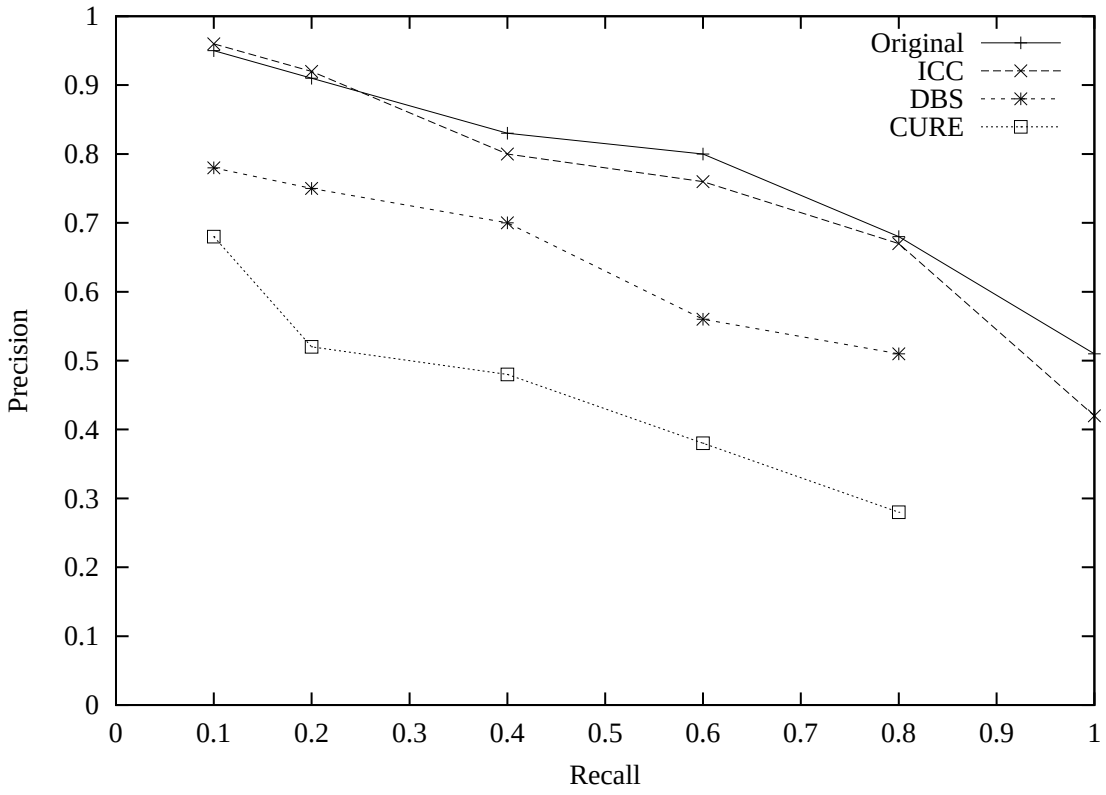


Figure 3.16: On Retrieval Effectiveness

Figure 3.16 shows the retrieval effectiveness of the various schemes. First, we note that the original scheme without clustering performs the best. This is expected since partitioning the database implies some data may be put into a partition that is not searched. Second, the proposed ICC performs nearly as well as the original scheme. We observe that the retrieval precision is slightly lower when recall is greater 0.3. Given the same precision, the original system has higher recall as it checked every single point. But the difference is not so much. At a reasonable recall of 0.6, the different is as little as 5%, with greater than 80% precision also. When recall is less 0.3, the precision is even higher. This is because ICC has the ability to separate some irrelevant new coming images when they are trying to find the right clusters. Third, ICC outperforms DBS and CURE by a wide margin. Because DBS has a better cluster's representation by density weighting than CURE, DBS achieved better performance. One possible reason for this that CURE and DBS are based on sampling and sampling techniques are not well suited for textual data. For textual data, once some information is lost, it' s hard to recover by sampling points, if not impossible. Finally, CURE and DBS cannot achieve 100% recall. This means that both do not cluster well for relevant images. Some are either clustered into other cluster or still in the 'noise box'.

These results clearly demonstrate that our ICC can produce quality clusters. It can cluster most similar points together and separate unrelated points way by split. By applying ICC, we still can achieve high precision with high recall.

Effect on Retrieval Efficiency

The main goal of clustering is to improve the retrieval efficiency at the expense of retrieval effectiveness. Figure 3.17 shows the results on the retrieval efficiency.

As shown in Figure 3.17, the original scheme (without clustering) performs poorly

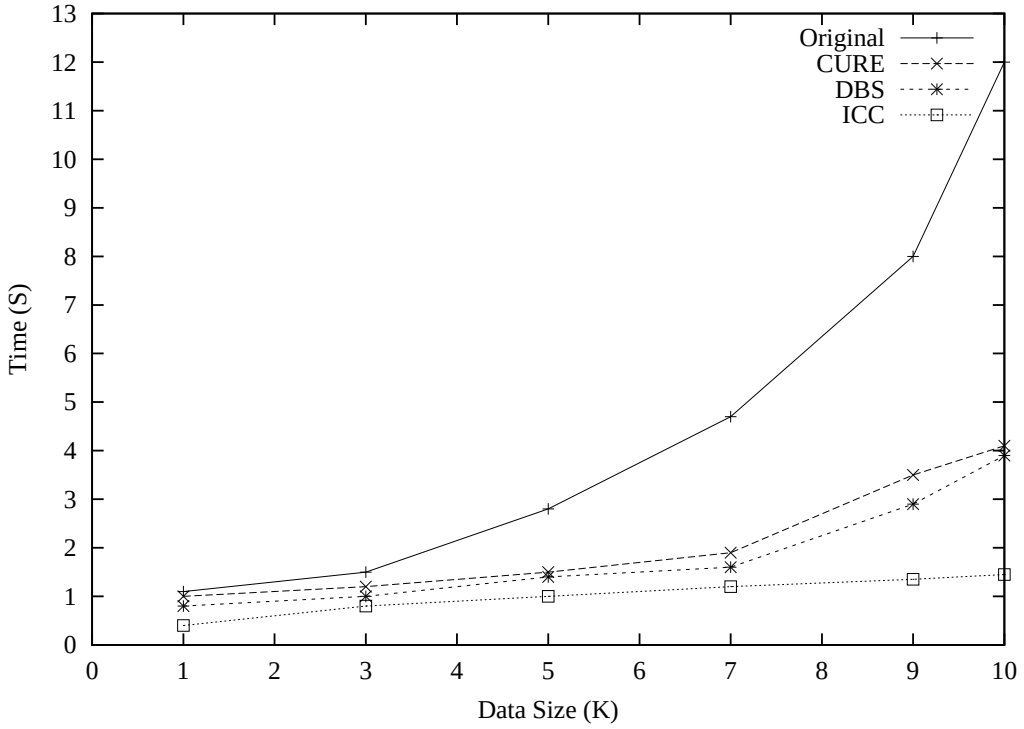


Figure 3.17: On Retrieval Efficiency

in terms of efficiency. This is expected since the scheme examines the entire database, while the cluster-based scheme only examines some relevant partitions. We also observe that both DBS and CURE are not as efficient as we expected. There are several possible reasons. First, both are quadratic time complexity to N. To scan the whole database, the complexity is O($N^2 * L^2 * D^2$). While for ICC, it's O($N * L^2 * D^2 * (ln(C) + U)$). With the data size increasing, the difference between ICC and the other schemes widens as (U+ln(C)) becomes smaller and smaller compared to N. Second, there is no automatic control on cluster's size and cluster number in CURE and DBS. There is a danger to cluster many images into a single cluster if they are from the same website. For example, if most images are from BBC websites, both methods may probably cluster them into a single group. In fact, this group may contain several sub- groups about sports, nature, entertainment, and so on. To

confirm what we thought, we explore the inner clusters in the database resulting from these two methods. There are two extremely large clusters regarding BBC and National University of Singapore. Whenever searching these two, it takes longer time than ICC. Due to the split operation with well control on cluster's upper bound, ICC can always split extremely large clusters into smaller ones after identifying its sub-shapes.

To summarize, ICC speeds up the original retrieval significantly. The larger the data set, the greater is the gain. From Figure 3.16 and Figure 3.17, we have reason to argue that our ICC can perform very well in WWW image retrieval, both in effectiveness and efficiency.

3.6 Extended ICICLE for Multiple Features

In Section 3.4, we have seen the proposed semantic-based retrieval system - ICICLE, for WWW images, by considering image's text description only. It is also interesting to include image's low-level or visual features, such as shape, color, and texture, to provide more functionalities for users to query on any type of features that is of user's interest. Next, we present an extended ICICLE for a generalized WWW image retrieval system in Figure 3.18 by considering image's multiple features.

Similarly to ICICLE, the extended ICICLE system also consists of five basic components: a Web Crawler, a Query Model Generator, an Image Search Engine, a Feedback Model Generator, and an Indexing/Clustering Module.

The Web Crawler that operates in the background automatically searches the WWW for pages with embedded images. The crawler analyzes the image and its HTML document to generate its multi-features representation model. It then loads the images with their representations models into the database. For fast database retrieval, Indexing/Clustering Module builds effective indices or clusters on top of the

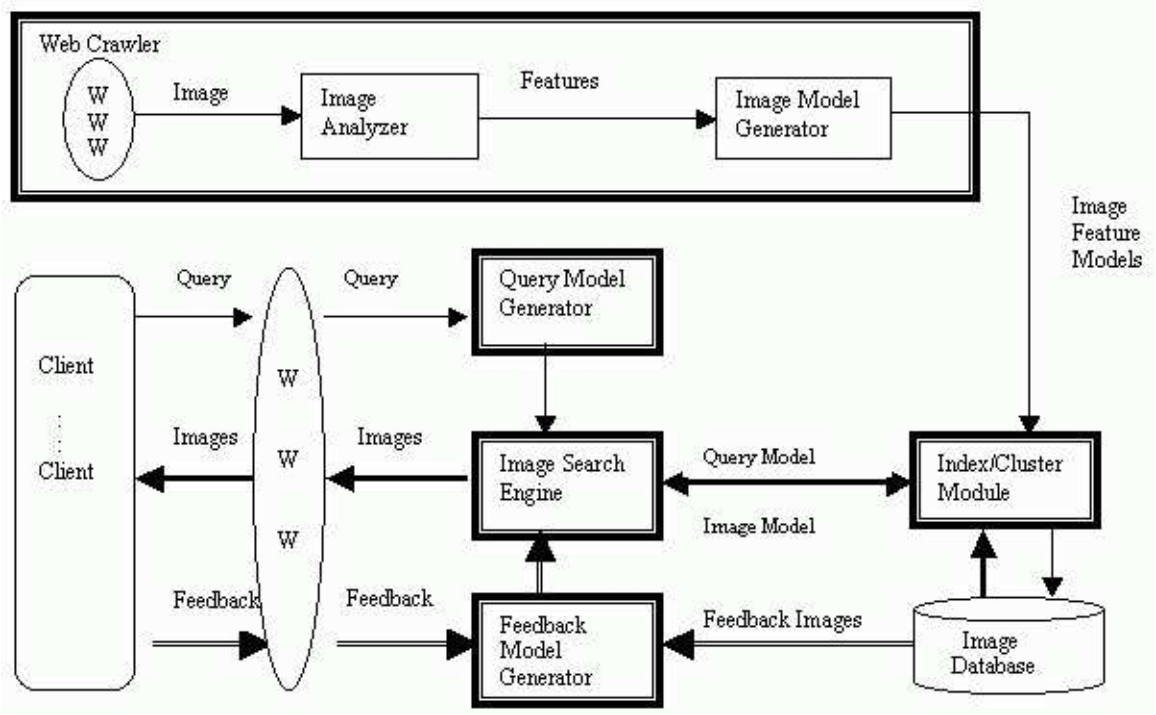


Figure 3.18: Extended ICICLE system structure in client-server form

images models. Whenever a query comes, the Query Model Generator transforms a user query to a query representation model (multiple features). Image Search Engine then passes the Query Model to Indexing/Clustering Module to facilitate the retrieval of relevant images. The Image Search Engine compares the Query Model against the Image Models returned from Indexing/Clustering Module, then integrate the similarities from all features and finally returns top related images. The images are displayed in order of decreasing degree of similarity. Finally, the Feedback Model Generator is used to generate extended query from the user query and the feedback images in the same way as ICICLE.

3.7 Implementation of Extended ICICLE

We have implemented the extended ICICLE under UNIX operating system. The Web Crawler works in background and collects meaningful images into database.

It is separated from other modules. We implemented the crawler in JAVA. Three image features are extracted from the collected images. The image's text feature is represented by Weight ChainNet model. As for visual features, color histogram feature and wavelet feature are considered. We use the 159-dimensional LUV color histogram. Wavelet features describe an image's shape, texture and location information in a single representation. Here we truncate the 64 most dominating wavelet coefficients as an image's visual feature. We use Daubechies' wavelet[58] to extract wavelet feature. As a result, color histogram and wavelet features are in 159- and 64-dimensionality respectively. All other modules are implemented in C/C++. The graphic user interface is designed in HTML form. And the communication between the sever and users is connected by CGI. Users may initialize the searching by either keying in text or selecting one of sample images as a query. From the returned images, users have the flexibility to consider any subset of three features and possible integration techniques. Furthermore, users may select any subset of returned images to perform feedback search for more accurate results.

As discussed in Related Work, indexing very high-dimensional points and multiple high-dimensional points are big challenges in database community. To provide efficient database support for extended ICICLE which considering multiple features, effective indexing techniques have to be proposed to tackle the 'dimensionality curse'. In the next chapters, we focus on the Indexing/Clustering Module and introduce three novel indexing techniques in order to indexing image database efficiently.

3.8 Summary

In this chapter, we have presented several techniques to solve the issues involved in an effective and efficient WWW image search engine. We first introduced a new model to represent the content of images embedded in WWW pages. The proposed

Weight ChainNet model combines different types of lexical chains obtained from the surrounding text of an image. Our experimental study showed that the approach can be used as an effective means to represent image semantics. We also proposed two novel feedback mechanisms. In particular, the semantic integration and differentiation method returned more accurate results than semantic accumulation with higher recall.

Moreover, we have also presented a new incremental clustering algorithm ICC for the increasingly growing large database collection of WWW images. Our experiments showed that ICC can produce quality clusters, and can adapt the cluster size and cluster number dynamically. Without looking at the actual data points, ICC can identify the sub-clusters within a large cluster by checking the cluster representative's property. It can also handle temporary noise well by using a special 'noise box' from which new clusters can be generated. To improve efficiency, ICC employs a summarization step called Vertical and Pyramidal Summarization Tree. VP-ST starts from all the data points in the cluster, and finally converges all summarized points into single representative of the cluster. Our experiments indicated that this vertical and pyramidal technique provides quality representative, especially compared with random sampling. To further speed up searching right cluster, the clusters' structure is in Hierarchical ChainNet Summarization tree. A prototype system, called ICICLE, that employs the proposed models have been deployed. ICICLE is then further extended to include multiple visual image features.

Chapter 4

Indexing High-dimensional Image Feature

4.1 Introduction

Image retrieval involve high-dimensional data which represent image's features. Indexing high-dimensional data has been an area of active research for a long time and many indexing techniques have been proposed [61]. However, the performance of these indexes attempting to scale up to high dimensions degrades rapidly with increasing dimensionality [9].

One approach to minimizing the effect of 'dimensionality curse' is to reduce the number of dimension of the high-dimensional data before indexing on the reduced dimension [42, 13]. Data is first transformed into a much lower dimensional space using dimensionality reduction methods and then an index is built on it.

Transforming data from a high-dimensional space to a lower dimensional space without losing critical information is not a trivial task. In this chapter, we propose a dimensionality reduction technique called *Multi-level Mahalanobis-based Dimensionality Reduction* (MMDR) for indexing based on the following two observations. First, elliptical shaped (correlated) clusters are more suitable for dimensionality reduction than spherical shaped clusters. Second, we observe that certain level of the

lower dimensional subspaces may contain sufficient information for correlated cluster discovery in the high-dimensional space. In the MMDR, *Principal Component Analysis*(PCA) [32] is employed to find the lower dimensions for dimension reduction. Most of the information in the original space can be condensed into a few dimensions along which the variances in the data distribution are the largest. We make use of the *Mahalanobis distance* (MahaDist) in our approach instead of the standard well-known L-norm distance functions.

Mahalanobis distance could be applied to find ellipsoidal correlated data, by taking local elongation into account. Instead of equally treating all values, MahaDist weights the differences by the range of variability in the dimension of the data points. It weights the variation along the axis of elongation less than that in the shorter axis of the ellipse. It can be shown that the surfaces on which MahaDist is a constant are ellipses.

Euclidean distance-based clustering algorithms that use Euclidean distance values to partition data are not meant to discover elliptical shape since the clusters identified are in circular shape. Figure 4.1 illustrates two clusters, one obtained using Euclidean distance and the other obtained by Mahalanobis distance. Point A is a valid point and point B is a noise in the cluster of the circle if Euclidean distance is employed. However, in terms of Mahalanobis measurements, point B has a substantially smaller distance to the centroid than point A since it lies along the direction of the group that has the largest variance. Thus point A is a noise while point B is valid. Therefore, while Euclidean distance based algorithms produces circular subsets as shown in Figure 4.1, Mahalanobis distance based algorithms will produce elliptical clusters where data points are well correlated and more natural for dimensionality reduction, as dimensions with large variance of data are kept and dimensions with small variance of data are eliminated.

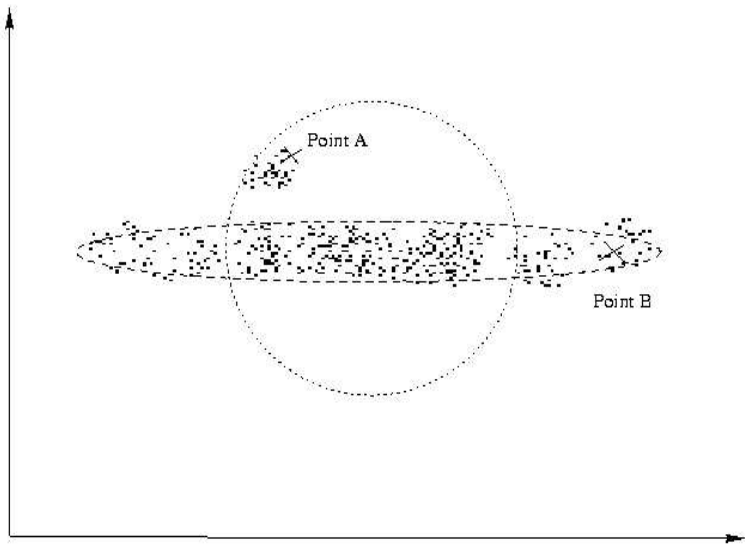


Figure 4.1: Mahalanobis vs. Euclidean

Based on multi-level low-dimensional projections produced by PCA and the Mahalanobis distance function, the MMDR can quickly identify highly correlated elliptical clusters. After the dimensionality reduction, each cluster of data is in a different axis system. Instead of creating one index for each cluster, we build one index for all the clusters for K nearest neighbor (KNN) queries. We extend a recently proposed B^+-tree based index - iDistance[61, 62], to index the data projections from the different reduced-dimensionality spaces. The extended iDistance allows us to index data points from different axis systems in a single index efficiently. Performance studies using real and synthetic datasets were conducted to evaluate the effectiveness and precision of the technique. The results show significant performance gain over an existing method [13]. Experiments on datasets with very high dimensionality (up to 200 dimensions) and datasets with dynamic insertions show that the proposed method is scalable in terms of both size and dimensionality and is able to adapt to dynamic insertions

The rest of this chapter is organized as follows. In Section 4.2, we provide the

definitions for using Principal Component Analysis and Mahalanobis distance in dimensionality reduction. We present our MMDR algorithm and its variant in Section 4.3. In Section 4.4, we propose an extended iDistance for indexing data points in reduced-dimensionality spaces and for handling dynamic insertions. Experiments are presented in Section 4.5 and conclusion is drawn in Section 4.6.

4.2 Definitions

In this section, we provide the basic definitions.

Definition 3.1 Ellipticity

Ellipticity(e) is the deviation of an ellipse or an ellipsoid from the form of a circle or a sphere, which is the ratio of the difference of the two sub-axes to the minor axis.

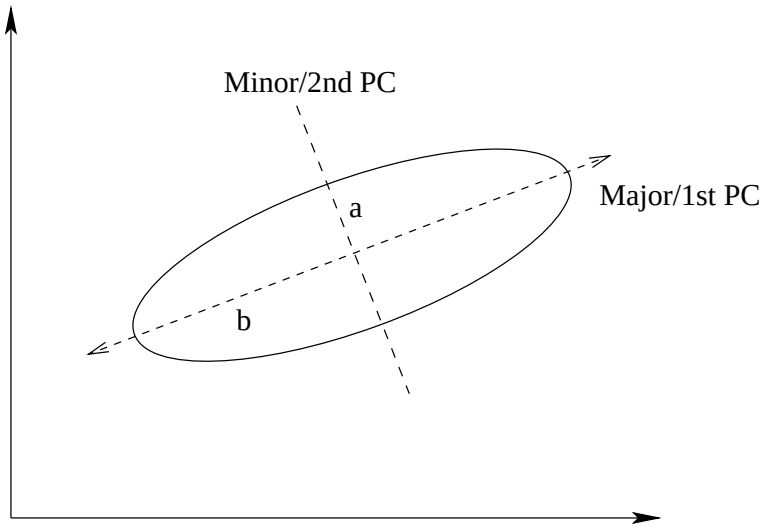


Figure 4.2: Illustration of Ellipticity

$$e = \frac{b - a}{a}$$

where b is the radius along the major axis and a is the radius along the minor axis, as shown in Figure 4.2.

Assuming all the points are clustered inside the ellipse. To reduce the dimensionality, all the points are projected onto the major axes. Thus, the minor axes can be eliminated. Obviously, the larger the e is, the more effective dimensionality reduction can be obtained. When e =0, the data points form a circle, and the dimensionality reduction technique becomes ineffective.

Definition 3.2 Mahalanobis Distance

The *covariance* of data in two feature spaces measures their tendency to vary together. In a multi-dimensional space, the variance measures the relative 'radius' of a cluster along each dimension, and the covariance indicates the orientation of the cluster. Both the variance and the covariance co-determine the shape of the cluster. Collecting them together, we get the covariance matrix **C**. Now let us look at the distance function called *Mahalanobis Distance* by using the inverse of covariance matrix.

Given a cluster centred at O, the *Mahalanobis Distance* between a point P and O is given as follows:

$$MahaDist(P,O) = (P-O)^T C^{-1}(P-O)$$

where **C** is covariance matrix describing cluster's shape.

From Mahalanobis Distance, we get a normalized measure: *Normalized Mahalanobis Distance*.

$$MahaDist_n(P,O) = \frac{1}{2}(d\ln(2\Pi\cdot \mid C \mid) + (P-O)^T C^{-1}(P-O))$$

where d is the dimensionality, Π is the trigonometric number 3.14 and $\mid C \mid$ is the determinant of **C**. Notice that given a spatial displacement between a point and an ellipsoid, the standard Mahalanobis Distance tends to be smaller for long clusters with large covariance matrices than that for small clusters. With standard Mahalanobis

Distance, the larger cluster will keep increasing in size and eventually overwhelm the smaller clusters. Normalized Mahalanobis Distance avoids such situation [55].

Definition 3.3 Multi-level Projections

Principal Component Analysis (PCA) [32] examines the variance structure in the dataset and determines the directions along which the data exhibits high variance. The first principal component is the eigenvector corresponding to the largest eigenvalue of the dataset's covariance matrix $\mathbf{C}$, the second component corresponds to the eigenvector with the second largest eigenvalue and so on. It is interesting to note that the Principal Components in PCA are just the eigenvectors of the covariance matrix in Mahalanobis Distance which describes the dataset's shape. An example is shown in Figure 4.3, where the preserved dimension is the first principal component, and the eliminated dimension is the second principal component. In dimensionality reduction, given a point P in a dataset, it has two projections. One is the projection on the preserved subspace P' that we are interested in; the other is the projection on the eliminated subspace P''. The d_r-dimensional projection P'_{d_r} can be defined as:

$$P'_{d_r} = P \cdot \Phi_{d_r}$$

where Φ_{d_r} represents the matrix containing 1^{st} to d_r^{th} principal components. Change d_r with different value, we can generate multi-level projections of the data for cluster discovery purpose in *Multi-level Mahalanobis-based Dimensionality Reduction* algorithm.

Definition 3.4 Projection Distance

From the above two projections, $ProjDist_r$ measures the distance from P to P' and $ProjDist_e$ measures the distance from P to P'' on the eliminated subspace. More

specifically, $ProjDist_r$ is the information lost from original representation P to its reduced d_r-dimensional representation P'. $ProjDist_e$ is the information retained. Figure 4.3 illustrates the two projection distances. In the following paragraphs, $ProjDist$ represents $ProjDist_r$.

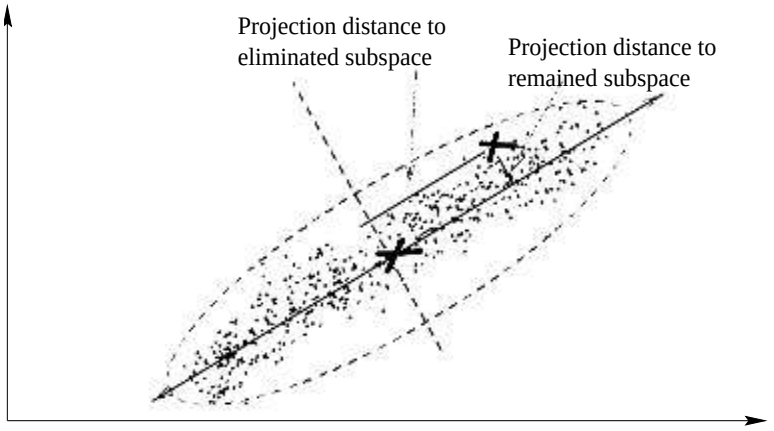


Figure 4.3: Two projection distances

Based on the above two projection distances, we extend the definition of **ellipticity** to multidimensional space as:

$$e = \frac{Max(ProjDist_e) - Max(ProjDist_r)}{Max(ProjDist_r)}$$

where $Max(ProjDist_e)$ is the radius along the remained subspace, and $Max(ProjDist_r)$ is the radius along the eliminated subspace. The cluster's Mahalanobis radius r is $Max(ProjDist_r)$. For dimensionality reduction, the larger the *ellipticity* value, the more effective dimensionality reduction can be performed.

Definition 3.5 Mean $ProjDist_r$ Error (MPE)

Mean $ProjDist_r$ Error is defined to be the average representation error when points are mapped from original space to eliminated subspace.

$$MPE = \frac{\sum_{i=1}^{N} ProjDist_r(P_i, O)}{N}$$

Symbols	*Descriptions*	*Value*
N	Data Size	
d	Original Dimensionality	
d_r	Optimal Dimensionality	
s_dim	Subspace Dimensionality	
e	Ellipsoid's Ellipticity	
r	Mahalanobis Radius	
C	Covariance Matrix	
$ProjDist_r$	Dist to remained subspace	
$ProjDist_e$	Dist to eliminated subspace	
MPE	Mean Projection Error	
σ	Outlier Set	
β	$ProjDist_r$ Threshold	0.1
MaxMPE	Max MPE Allowed	0.05
EC	Elliptical Cluster	
MaxEC	Max EC allowed	10
MaxDim	Max Remained Dim allowed	20
ε	Data Stream Percentage	0.005
ξ	Outlier Percentage	0.005
k	Num of IDs in lookup table	3

Table 4.1: A Table of Symbols and default values in Chapter 4

Table 4.1 gives a summary of the symbols in this chapter and their respective description with default values used in experiments.

4.3 Multi-level Mahalanobis-based Dimensionality Reduction (MMDR)

4.3.1 MMDR Algorithm

The Multi-level Mahalanobis-based Dimensionality Reduction (MMDR) algorithm, which is outlined in Figure 4.4, consists of two major steps, namely: *Generate Ellipsoid* and *Dimensionality Optimization*.

In *Generate Ellipsoid*, we recursively apply multi-level projections from low to high dimensionality until ellipsoids are fully discovered. At each level, Mahalanobis distance is applied to detect possible ellipsoids. Any unqualified ellipsoid is passed to *Generate Ellipsoid* with a higher subspace dimensionality so that more information can be used for clustering. We adopt this divide-lower-before-conquer-upper approach based on the following observations. First, in high dimensional space, some dimensions may contain little information, which may not be very helpful when it comes to identify the cluster membership. Second, for the well-separated clusters in the subspace, their correspondences in the higher dimensional space are usually well separated because of the property of PCA. In our algorithm, MPE indicates how much information is lost during the projection process. It is used as the parameter to determine if the subspace projections carry enough information to reflect the shape of their correspondence in the original space.

The *Generate Ellipsoid* is invoked with a small subspace dimensionality - s_dim.

MMDR Algorithm

Generate Ellipsoid (GE)

```
Variable: ellipsoid_array, subspaces;
GE(data, d, s_dim)
1.    projections ← getProj(data, s_dim);
2.    semi_ellip ← ellip_k_means(projections,s_dim);
3.    // process each semi_ellips
4.    for each semi_ellip with size > 0
5.          semi_ellip_data ← restoreData(semi_ellip);
6.          semi_ellip ← getProj(semi_ellip_data, s_dim);
7.          MPE ← getMPE(s_dim);
8.          if MPE > MaxMPE and 2*s_dim > d
9.               GE(data, d, 2*s_dim);
10.         else
11.              add semi_ellip_data into ellipsoid_array
```

Dimensionality Optimization

```
12.   for each ellipsoid_array[i]
13.         d_r ← min(MaxDim, s_dim);
14.         MPE ← getMPE(d_r);
15.         while change of MPE < threshold
16.              d_r −− ;
17.              MPE ← getMPE(d_r);
18.         projections ← getProj(ellipsoid_array[i],d_r);
19.         for each projection
20.              ProjDist ← getProjDist();
21.              if ProjDist ≤ β
22.                   add it to this subspace;
23.              else
24.                   add it to noise set
```

Figure 4.4: MMDR Algorithm

In line 1 of Figure 4.4 the low dimensional projections are produced from the original d-dimensional space, followed by elliptical k-means clustering [55] in this low dimensional subspace, line 2. The data are then partitioned into semi-ellipsoids at s_dim-dimensional subspace. We call this $semi - ellipsoid$ since we have not decided yet whether it properly indicates the shape of its correspondence in the original space. From line 3 to line 11, each semi-ellipsoid is handled individually. For each semi-ellipsoid discovered above, its corresponding shape is restored in the original dimensional space (line 5), and its local s_dim-dimensional subspace is generated(line 6). The newly produced projections are local to individual semi-ellipsoid and different from the projections produced in line 1. At line 7, the MPE to s_dim-dimensional subspace is computed.

If a semi-ellipsoid has smaller MPE than the maximum error allowed, it suggests that the s_dim-dimensional subspace can approximately represent its original data. Otherwise, there are two possible reasons for the big MPE. First, it could be due to the overlap of several clusters in the subspace such that each point did not project to its local subspace. Higher subspace dimensionality should be retained in order to distinguish each cluster. Second, though it is a single cluster, the s_dim could be too small for a subspace to represent original dimensional data. To further discover ellipsoids in each semi-ellipsoid, we increase the s_dim twofold without losing generality and recursively call *Generate Ellipsoid* (line 9). Therefore, the semi-ellipsoid is repeatedly partitioned locally. This step produces possible ellipsoids. It should be noted that the process of discovering ellipsoids in the subspaces is the first step of dimensionality reduction, where the remained subspace dimensionality of each ellipsoid at this stage is their respective s_dim, and further dimensionality optimization is performed in the next step.

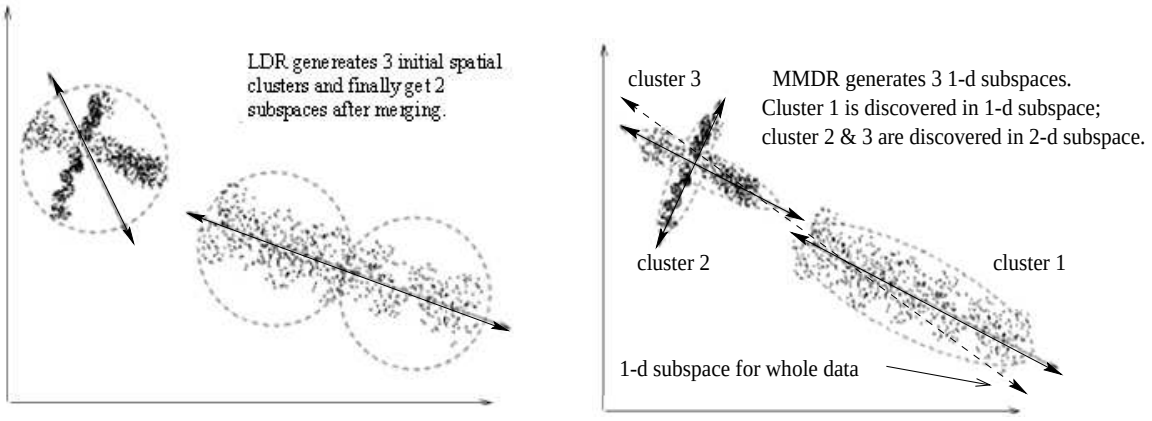
Since the above step produces possible ellipsoids in their respective s_dim-dimensional

subspace, and ellipsoids are effective for dimensionality reduction, the *s_dim* of the subspaces discovered in *Generate Ellipsoid* can be further reduced (it should be noted that each ellipsoid may correspond to a different *s_dim* value). That is, if the change of MPE is less than the pre-set threshold, we decrease the dimensionality by 1 and the process is repeated till the above condition is false (line 15-17). The final dimensionality is treated as the 'optimal' one and denoted as d_r. The points are projected into this d_r-dimensional subspace (line 18). A threshold value β is employed to determine whether a point belongs to a cluster. If the projection distance on $(d-d_r)$-dimensional eliminated subspace for a point is greater than β, this point is taken as an outlier (line 23-24). Otherwise, it is classified as a member of the subspace (line 21-22).

The final output of the algorithm is a set of subspaces and outliers. Each subspace may have a different optimal number of reduced dimensions. The outlier set remains in the original space since its data points are not well correlated.

In a dataset, some clusters are elongated along certain directions and yet they are locally correlated. Such elongation may be detected in its lower dimensional subspaces. Given a 2-dimensional subspace as shown in Figure 4.5 projected from a higher dimensional space (say 4-dimensional). This 2-dimensional subspace can represent the original dimensional space with very little information being lost. The LDR technique [13] is able to discover correlated clusters on the original 4-dimensional space and produces two 1-dimensional subspaces as shown in Figure 4.5a. In order to partition the bigger shape cluster for dimensionality reduction, the clustering radius must be suitably large. However, this will result in smaller clusters being grouped together as one. Obviously, substantial information is lost for the smaller shaped clusters. The situation is even worse for small shaped clusters with a high density.

Figure 4.5b shows three 1-dimensional subspaces produced by MMDR algorithm. MMDR first projects the original dimensional space into 1-dimensional subspace, then



(a) Clusters generated by LDR using Euclidean distance

(b) Clusters generated by MMDR using Mahalanobis distance

Figure 4.5: LDR vs MMDR

elliptical k-means method partitions 1-dimensional projections of whole data into two partitions: cluster 1's 1-dimensional subspace and cluster 2 and 3's 1-dimensional subspace. After restoring cluster 1's 1-dimensional subspace back to original dimensional space and performing local 1-dimensional projections (line 5-6), MMDR detects that it is an ellipsoid since its MPE is small. The 1-dimensional subspace projected from clusters 2 and 3 overlaps heavily with the high MPE and thus its corresponding full dimensional shape/data are passed to *Generate Ellipsoid* by increasing the subspace dimensionality to be 2-dimensional. At the 2-dimensional subspace, these 2 ellipsoids can be discovered by the Mahalanobis Function. Dimensionality reduction is further performed in *Dimensionality Optimization* so that both can be reduced to 1-dimensional subspaces with less information lost than using LDR method.

In summary, MMDR has the following advantages. First, the ellipsoids can be effectively discovered at data's subspace level, rather than at the original space. Second, the ellipsoids are able to be discovered as soon as the shapes can be identified. Third, the cost to perform clustering using Mahalanobis distance can be reduced

dramatically since it is performed in the low dimensional subspace.

The cost of MMDR comes mainly from elliptical k-means method (line 2), which takes $O(Iter_{out} * Iter_{inn} * d\_sim^2 * N * MaxEC)$, where $Iter_{out}$ and $Iter_{inn}$ is the number of iterations for outer and inner loop respectively, d_sim^2 comes from distance computation. However, in MMDR, the input dimensionality s_dim is very small compared to the original d. The input data size N becomes smaller as s_dim increases, which leads to $Iter_{out}$ and $Iter_{inn}$ being reduced also. We further reduce the cost in the next subsection.

4.3.2 Optimization on Distance Computation

We note that the most time consuming step of the MMDR algorithm is the Mahalanobis distance computation between centroids and data points in elliptical k-means method. In this subsection, we reduce the computational cost by using the following techniques. The factor *MaxEC* can be reduced to a small number by avoiding computing all the distances between *MaxEC* centroids and a data point. We only need to re-compute the distance between the k most closest centroids which might change the membership of a point, where $k << MaxEC$. This is based on the following observations. First, if a data point is to be re-assigned to another cluster, that cluster is most probably the one with the closest distance except the current assignment. Second, in each iteration, only a small portion of data points might change their membership. As the converging process continues, the number of data points changing memberships decreases quickly. Third, some data points may never change their membership.

A lookup table is designed to store the k most closest centroids' IDs computed in the previous iteration for each data point. In the next iteration, only those centroids whose IDs are stored in the lookup table are taken to compute and find the closest centroid. A data point entry in the lookup table is updated only when its membership

is changed. By doing so, the factor *MaxEC* is removed from the overall cost.

To further reduce the cost for large datasets, we introduce one additional field called *Activity* to the lookup table to indicate how frequently a data point changes its membership. It records the number of iterations that a data point does not change its membership. If the value of *Activity* is larger than a threshold, we say this data point is *inactive*, otherwise *active*. *Inactive* data points need not make any further distance computation and re-assignment unless the number of clusters is changed. This reduces the value of N dramatically at each iteration. Assume that at each iteration, only $\frac{1}{Iter_{inn}}$ of the dataset change their memberships, the factor of N is replaced by $\frac{N}{Iter_{inn}}$. As the converging process continues, the number of points which change their membership decreases dramatically. Therefore, the time complexity becomes $O(Iter_{out} * d\_sim^2 * N)$. Comparing with the LDR's time complexity of $O(N * d^2 * MaxEC)$, MMDR has a smaller dimensionality of d_sim than d, but with a larger factor of $Iter_{out}$ than $MaxEC$.

4.3.3 Scalability for Large Datasets

Scalable MMDR Algorithm

1. Initialize the Ellipsoid Array;
2. Divide dataset into $\frac{1}{\varepsilon}$ data streams;
3. For each data stream
4. Pass it to *Generate Ellipsoids*;
5. Add newly generated ellipsoids into EA;
6. Form a new data stream using all centroids in EA;
7. Pass this new data stream to *Generate Ellipsoids*;
8. Re-assign point membership to the ellipsoids;
9. For each ellipsoid
10. Pass it to *Dimensionality Optimization*;

Figure 4.6: Scalable MMDR Algorithm

For a very large dataset which cannot be completely loaded into the main memory buffer, the data scan at each iteration is extremely expensive. To make the MMDR scalable for very large datasets, we divide the dataset into a number of data streams, which is defined as a sequence of data points read in order of indices, and we process one data stream at a time. We set the size of a data stream to be ε percent of the data size. A temporary array called Ellipsoid Array (EA) is created to store the ellipsoids' centroids generated for each data stream. Scalable MMDR loads a single data stream at a time and performs *Generate Ellipsoid* operation to generate small size ellipsoids. These small ellipsoids' centroids are stored in the Ellipsoid Array. After all the data streams have been processed, only the Ellipsoid Array is in the buffer. By calling *Generate Ellipsoid* on Ellipsoid Array, Scalable MMDR forms bigger size of ellipsoids by merging smaller ellipsoids whose centroids are stored in Ellipsoid Array. The detailed algorithm is outlined in Figure 4.6.

The size of data stream is much smaller than original data size N. Empirically, it is reasonable to expect that $Iter_{datastream} < Iter_{originaldata}$. Hence, the total time required to cluster $\frac{1}{\varepsilon}$ data streams of size $\varepsilon * N$ is generally less then the time required to cluster N data points.

4.4 Indexing Reduced Subspaces

After dimensionality reduction, the projections in reduced dimensionality subspaces have to be indexed using efficient indexes. Instead of using an index for each subspace, we want all the projections to be indexed in a single structure for ease of maintenance. We selected the iDistance [61] as our base index due to its efficiency and its B^+-tree base structure.

The design of iDistance was motivated by two factors. One, the triangular inequality relationships enable the (dis)similarity between a query point and a data point to be derived with reference to a chosen reference point. Two, data points can be ordered based on their distances to a reference point, and indexed based on such distance value. This enables one to represent high-dimensional data in a single dimensional space and use an existing B^+-tree. However, the iDistance has to be extended to index subspaces in different axis systems and handle dynamic insertion of data points.

4.4.1 Extended iDistance

The data partitioning strategy and reference point selection are straight forward, as the data partitions are determined by the MMDR algorithm and the centroid of each cluster is the ideal choice as the reference point. For each subspace (outliers as a subspace in its original dimensionality), all data points in subspaces are represented in a single dimensional space with reference to its centroid of cluster. This is achieved by the following mapping function:

$$y = i \times c + dist(P, O_i)$$

where the P is a data point in the subspace of i^{th} ellipsoid EC_i, and O_i is its centroid. $dist(P, O_i)$ is the distance function that returns distance between O_i and P. y is the index key for P. c is some constant to stretch the data range so that distance values are range partitioned based on reference points. That is, it serves to partition the single dimension space into regions so that points in the i^{th} cluster will be mapped to the range [i $\times$ c, (i+1) $\times$ c].

Extended iDistance employs three data structures:

- A B^+-tree is used to index the transformed single value points to facilitate speedy retrieval.

- An array is required to store the centroids and Principal Components of ellipsoids, and their respective nearest and farthest radius that define the subspace. This array is used for searching purpose.

- An array is required to store covariance matrices of ellipsoids, Mahalanobis radius, and the dimensionality retained. This array is used for the purpose of dynamic insertion.

To search for the K nearest neighbors of a query point q, the distance of the Kth nearest neighbor to q defines the minimum radius required for retrieving the complete answer set. Such a distance cannot be predetermined, and hence, an iterative approach that examines increasingly larger sphere in each iteration has to be employed.

The algorithm works as follows. Given a query point q, finding K nearest neighbors (NN) begins with a query sphere defined by a *relatively small* radius R around q. For each cluster EC_i, the query point is mapped into q_i, which is the projection of q on the i^{th} subspace.

Figure 4.7 shows an example with 3 clusters' max radius ranges in the different axis systems, where EC_1 is in XY plane, EC_2 in XZ plane and EC_3 in YZ plane. Here, for a query point q, its projection on 3 subspaces are q_1, q_2, and q_3 respectively. The shaded regions are the areas that need to be checked.

Searching in extended iDistance begins by scanning the auxiliary structure to identify the centroids whose data space (sphere area of cluster) overlaps with the query sphere defined by q_i and R. The search starts with a small global radius R for all subspaces, and step by step, the radius is increased to form a bigger query sphere.

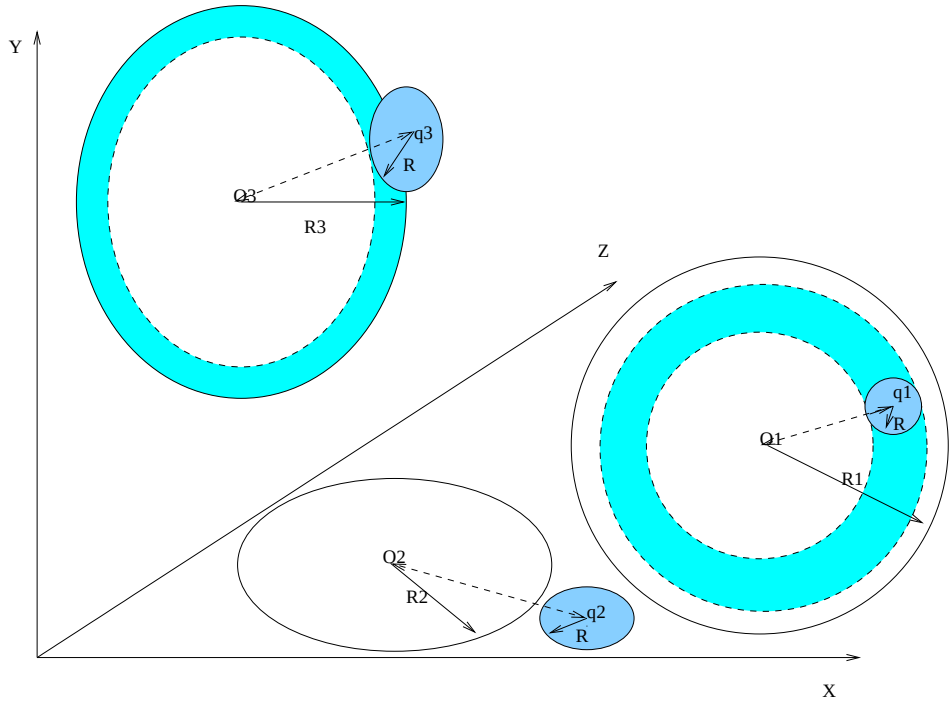


Figure 4.7: Searching for NN queries q_1, q_2 and q_3

For each enlargement, there are three main cases to consider.

- The data space EC_i contains q_i. In this case, we want to traverse the data space sufficiently to determine the K nearest neighbors. This is done by first locating the leaf node where q_i may be stored. Since this node does not necessarily contain points whose distance are closest to q_i compared to its sibling nodes, we need to search left and right (inward and outward of data space) from the reference point accordingly. This situation is illustrated by the subspace EC_1 and q_1.

- The data space intersects the query sphere. In this case, we only need to search leftward (inward) since the query point is outside the data space. This situation is illustrated by the subspace EC_3 and q_3.

- The data space does not intersect the query sphere. Here, we do not need to

examine the data space. This situation is illustrated by the subspace EC_2 and q_2.

The search stops when the distance of the K^{th} NN object to q is less than search radius R. The search is correct as the distance between image query point and data point always lower bounds the actual distance between the actual query point and data point in the original space. The searching subspace can be fast pruned by using triangle inequality property.

$$\|Q - P\| \geq \|Q_j - P_j\| \geq \|Q_j - O_j\| - \|P_j - O_j\| \geq \|Q_j - O_j\| - R_j$$

Where Q is query, P is original data point, Q_j is the projection in j^{th} subspace, P_j is the projection of P in j^{th} subspace, O_j is the reference point in j^{th} subspace, and R_j is the max radius in j^{th} subspace. $\|Q_j - O_j\| - R_j$ specifies the tightest searching bound for j^{th} subspace.

4.4.2 Handling of Dynamic Insertions

In the real world, the database is not static, and it is too costly for a dimensionality reduction algorithm to reanalyze the whole database after some insertions. It is granted that for a very large database, a single data point insertion should not greatly affect the effectiveness of clustering and dimensionality reduction. However, over time, the query accuracy and efficiency will be affected if the indexing mechanism is not adaptive to new data insertions.

To handle dynamic insertions, the MMDR algorithm is made to be adaptive to new insertions by splitting and merging subspaces if necessary. In the algorithm below, we treat a set of insertions as a batch insertion of a small dataset, which is a data stream described described in Section 4.3.

Dynamic MMDR Algorithm

```
1.   for each point P in data stream
2.          O ← nearestEllipsoid(P);
3.          dist_min ← minMahaDist(O);
4.          if dist_min ≤ O.r
5.              projection ← getProj(O.d_r);
6.              y ← getKey(O);
7.              insertIntoIndex(y);
8.              removeFromDataStream(P);
9.  subspaces ← MMDR(data stream);
10. for each subspace
11.         if merge(i)
12.             deleteFromIndex(i*c, (i+1)*c);
13.             form a new subspace with same ID - i;
14.         else
15.             Treat it as new subspace with new ID;
16. for each new subspace O_new
17.         for each point
18.             y ← getKey(O_new);
19.             insertIntoIndex(y);
```

Figure 4.8: Dynamic MMDR Algorithm

For each data point P in data stream, we first get the ellipsoid with minimal MahaDist to P and denote it as O (line 2-3). If this distance is less than O's Mahalanobis radius - r, get its projection in O's subspace (line 5), followed by mapping it to indexing value (line 6) and inserting it into the index (line 7), then remove it from data stream. Next, the updated data stream is passed to MMDR to generate subspaces (line 9). For each subspace, if it has same elongation and intersects with an existing subspace i, delete all the entries whose indexing values are in the range of $[i * c, (i + 1) * c]$ (line 12) and form a new subspace with same ID - i (line 13). Otherwise, treat this subspace as a new subspace with a new ID (line 15). Finally for each new subspace (line 16-19), map each data point into indexing value with corresponding subspace, then insert into the index.

One important step in the above algorithm is to merge two subspaces when necessary. The following condition must be satisfied in order to merge two subspaces: their ellipsoids have the same elongation and intersect with each other. Figure 4.9 illustrates such scenario. In Figure 4.9, we only indicate the circumscribing ellipse of the cluster and r is the Mahalanobis radius. If the MahaDist of two centroids of ellipsoids is not greater than the sum of their Mahalanobis radius, these two ellipsoids intersect. In this case, two ellipsoids are merged as a new ellipsoid. Based on the new centroid, we re-map the points into indexing values with the existing ellipsoid's ID, followed by standard insertion operation.

It is important to note that the algorithm does not require us to rebuild the whole tree. Instead, it only affects the partitions where merging is performed.

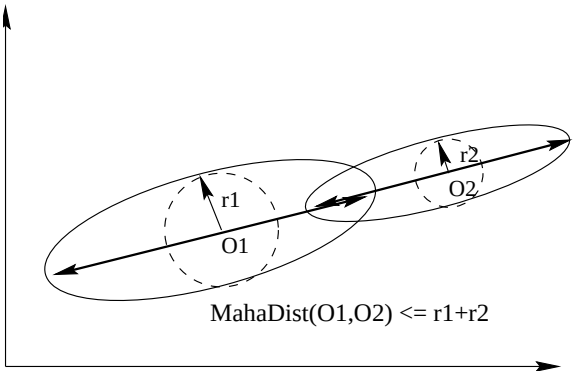


Figure 4.9: Two ellipsoids intersect with same elongation

4.5 Performance Study

In this section, we present the performance study to evaluate the effectiveness of MMDR and the efficiency of extended iDistance. For the experiments, we use the default values as shown in Table 4.1, and all experiments were done with Ultra-10 SunOS 5.7 processor (333 MHz CPU and 256 MB RAM).

We have two categories of test data.

- Real life datasets: We have two sets of real life datasets. One consists of 64-dimensional Daubechies' wavelet[58] features extracted from 73,715 WWW images randomly crawled from over 40,000 websites by ICICLE system. Wavelet features describe an image's shape, texture and location information in a single representation. Here we truncate the 64 most dominating wavelet coefficients as an image's visual feature. The other consists of 64-dimensional color histogram extracted from 70,000 color images from Corel Database, used in LDR[13].

- Synthetic datasets: We have four sets of synthetic datasets. One small synthetic dataset contains 100,000 points in 64-dimensional space. Three large synthetic datasets with 1,000,000 points are in 50-, 100-, and 200-dimensional spaces respectively. For each synthetic dataset, we use the algorithm 4.10to generate

Parameters	*Descriptions*
N	Data Size
d	Original Dimensionality
s_dim	Subspace dimensionality
EC	Number of elliptical clusters
EC_size	Size of elliptical cluster
s_r_dim	Starting remained dimension for EC
variance_e	Variance for eliminated subspace
variance_r	Variance for remained subspace
lb	Lower bound value for EC

Table 4.2: Table of input parameters and description

correlated clusters in different subspaces with different distensibilities. Each subspace has different size, orientation and ellipticity.

In order to generate the local correlated datasets, we use the algorithm outlined in Figure 4.10 to generate different clusters in different subspaces with different orientations and distensibilities retained. Table 4.2 gives the descriptions of input parameters.

Generate Correlated Dataset (GCD)

input: N,d,EC,EC_size[EC],s_dim[EC],s_r_dim[EC],
variance_e[EC],variance_r[EC], lb[EC]
Output: datasets[EC]
Algorithm:
1. for i from 0 to EC-1 do
2. for j from 0 to EC_size[i]-1 do
3. for k from 0 to s_remained_dim[i]-1
4. datasets[i][j*d+k]= gen_float(lb[i], variance_e[i])
5. for k from s_r_dim[i] to s_r_dim[i]+s_dim[i]-1
6. datasets[i][j*d+k]= gen_float(lb[i], variance_r[i])
7. for k from s_r_dim[i]+s_dim[i] to d-1
8. datasets[i][j*d+k]= gen_float(lb[i], variance_e[i])
9. rotate datasets[i] to be arbitrarily oriented

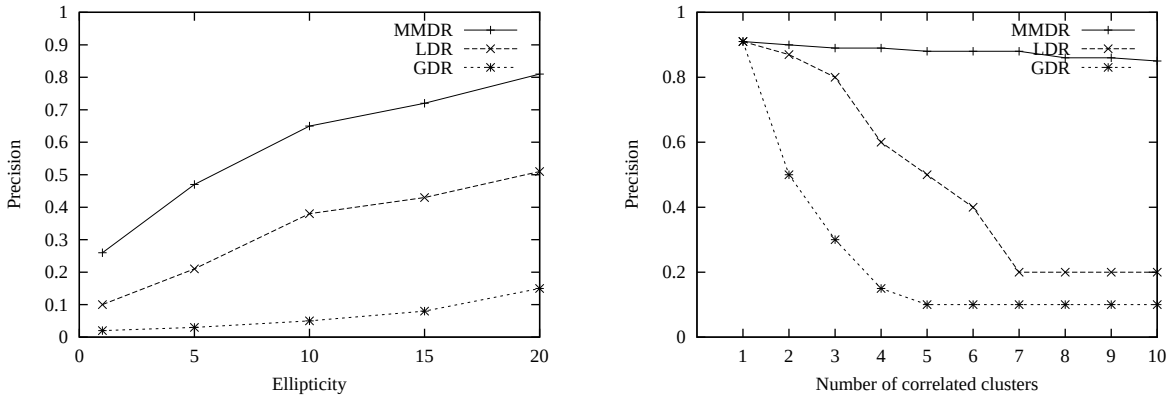
Figure 4.10: Synthetic Datasets Generation

In this algorithm, array s_dim[i] contains the dimensions which should be remained for each cluster.We can randomly choose which dimension should be retained. For simplicity, we make remained dimensions continuous starting with s_r_dim[i]. For example, if s_r_dim[i]=6, then the remained dimensions for i^{th} cluster starts from 6^{th} to $(6+s\_dim)^{th}$ dimensions. Specifying the different values for each cluster allows each reduced subspace in different axis systems. Method gen_float() will return a random float value in [lb,lb+variance]. It can also return a value based on other distribution functions, such as Zipfian. For each cluster, we also specify their different lower bound values, which can be used to control the positions of centers of each cluster together with its variance. Along each of the remaining s_dim[i] dimensions, we assign a randomly chosen value falling in range of [lb[i], lb[i]+variance_r[i]] to all the points in the cluster. Along each of the reduced (d-s_dim[i]) dimensions, we assign a randomly chosen value falling in range of [lb[i], lb[i]+variance_e[i]] to all the points in the cluster. The ratio between variance_r[i] and variance_e[i] in fact specifies the ratio between the energy carried by remained and reduced dimensions for each cluster, or the degree of correlation/ellipticity. Both values can be adjusted for different clusters in order to have different level of correlation. To make the subspace arbitrarily oriented, we can generate a random orthonormal rotation matrix (generated using MATLAB) and rotate the cluster by multiplying the data matrix with the rotation matrix.

We used 100 queries to obtain the mean precision and query cost on 10NN, and L_2 distance was used for searching (Note that the Mahalanobis distance is used for discovering intrinsic ellipsoids, not for searching). The query precision is defined as follows:

$$Precision = \frac{R_{d_r} \bigcap R_d}{R_d}$$

where R_d and R_{d_r} are the results respectively returned from the original space and



(a) Effect of ellipticity

(b) Effect of number of clusters

Figure 4.11: Effect on precision

reduced subspaces.

4.5.1 Query Precision

Here dimensionality reduction methods serve for the purpose of efficient indexing. However, they are lossy in nature. When a dimensionality method tries to reduce more, it may cause bigger loss of information and hence query precision. The query precision is also affected by correlation between data points and number of correlated clusters. Here we use the small dataset with 100,000 points.

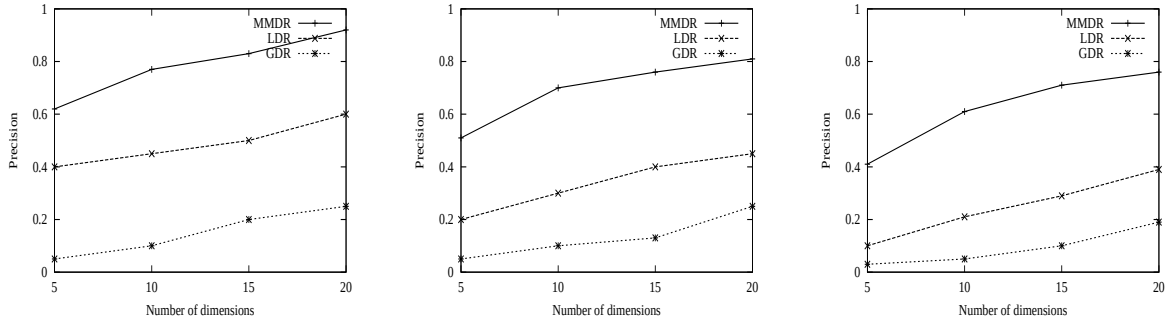
Figure 4.11a shows the query precision with respect to increasing ellipticity. As we can see, the MMDR method performs much better than the LDR and GDR methods. The GDR method can achieve at most 15% of precision as the dataset is not globally correlated. As ellipticity decreases, LDR drops faster than MMDR. Obviously, less correlation has more negative effect on the query precision of LDR then MMDR. In the next experiment, we varied the number of correlated clusters to test its effect on query precision. The results in Figure 4.11b show that all MMDR, LDR and GDR perform equally well when there is only one correlated cluster. But as the

number of correlated clusters increases, the MMDR is able to locate all correlated clusters effectively and maintains its query precision. However, the query precision of the LDR drops rapidly, and so does the GDR method. It indicates that when clusters intersect and have different ellipticities and scales, LDR can not discover all of them. As more such clusters exist, LDR performs worse. In contrast, the MMDR can discover the intrinsic number of correlated cluster based on Mahalanobis distance and thus is independent of the number of correlated cluster.

To see the effect of the number of eliminated dimensions on the effect of query precision, we conducted experiments using the small synthetic dataset and two real datasets. In this experiment, we set the maximum remained subspace dimensionality $MaxDim$ to be 20. Figure 4.12 presents the effect of the number of dimensions retained after dimensionality reduction on the query precision on two datasets. All the three methods show increasing precision as the remained dimensionality increases for three datasets. MMDR achieves much higher precision. As shown in Figure 4.12a for the synthetic dataset, at 20 dimensions, LDR only can achieve at most 60% of precision. and GDR cannot achieve more than 25% of precision due to uncorrelated property. Figure 4.12b shows the effect of retained dimensionality on the query precision using the color histogram dataset. It is interesting to note all three methods are not performing as well as before. Nevertheless, the MMDR method performs the best and is least affected. The higher precision obtained by the MMDR method confirms two important observations. First, there exist some local elongated clusters. Second, some intrinsic local elongated *shapes/correlations* cannot be detected by LDR. Compare to the synthetic dataset (Figure 4.12a), the precision of the methods on color histogram dataset are much worse. One reason could be that the real dataset may have clusters that are highly uncorrelated. Too many outliers may be another reason. This is possible, as for each image in the real dataset, the color histograms

tend to be very skewed towards a small set of colors, with many attributes being 0. Figure 4.12c shows the effect of retained dimensionality on the query precision using the wavelet feature dataset. It has the similar trend as Figure 4.12b, but with lower precisions than color histogram dataset for all three methods. The reason is possibly that higher uncorrelated property exists in our WWW images. Since our images were randomly crawled from WWW, their wavelet features may not be well correlated in some clusters. However, the gap between LDR and MMDR becomes even larger for this dataset.

The above experiments confirm that the MMDR method is a much more effective dimensionality reduction technique in correlated environments with lower loss of distance information, as it can achieve the better reduction performance with higher precision, which should lead to faster searching and retrieval.

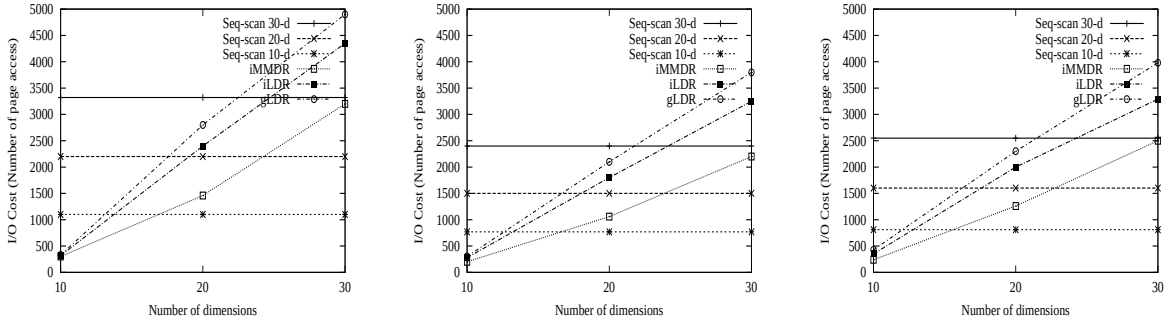


(a) Synthetic data (b) Color histogram (c) Wavelet features

Figure 4.12: Effect of dimensionality on query precision

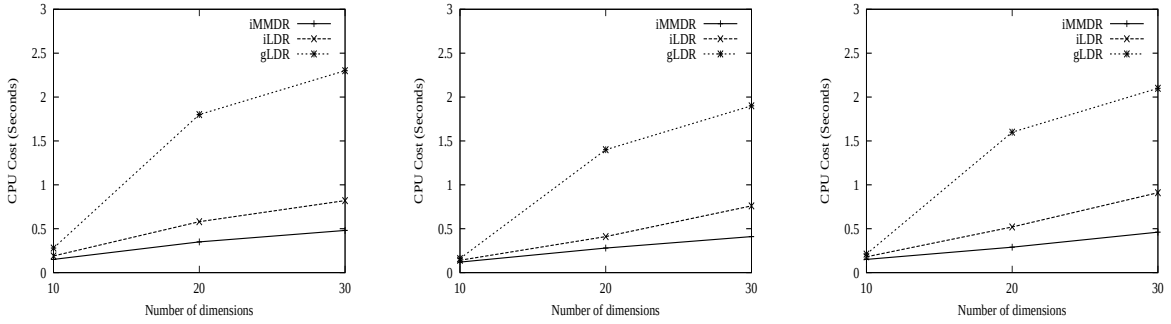
4.5.2 Query Efficiency

In this experiment, we examine the query performance of the index methods on reduced dimensionality data points. Note that the final purpose of performing effective dimensionality reduction by using MMDR is to improve the query performance, as it



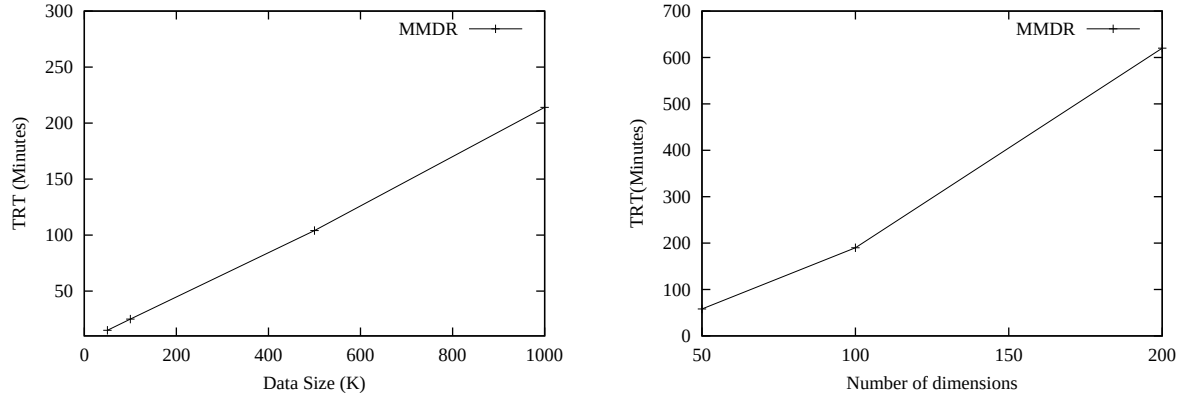
(a) Synthetic data (b) Color histogram (c) Wavelet features

Figure 4.13: Effect of dimensionality on I/O cost



(a) Synthetic data (b) Color histogram (c) Wavelet features

Figure 4.14: Effect of dimensionality on CPU cost



(a) Effect of data size (b) Effect of dimensionality

Figure 4.15: Effect on total response time

is well known that existing multi-dimensional indexing structures are not be able to index very high (30 or greater) dimensional data space. Here we have three indexing schemes to compare: extended iDistance on MMDR data (iMMDR), extended iDistance on LDR data (iLDR) and Global indexing method [13] on LDR data (gLDR). The Global indexing method makes use of one Hybrid tree [14] for each cluster, and maintains the information about each cluster and index in an array. Here we use the same datasets as in the last sub section.

Figure 4.13 shows the I/O cost for three indexing schemes, and sequential scan in reduced subspaces, when the subspace dimensionality varies from 10 to 30. Figure 4.13a shows that for synthetic dataset, as the dimensionality increases, the iMMDR has much lower I/O cost than the iLDR, which confirms that *a more effective dimensionality reduction method leads to an overall improved query efficiency.* We also notice that the gLDR is worse than the iLDR, and when the dimensionality reaches 20, its cost is higher than that of direct sequential scan. The extended iDistance is more efficient in terms of I/O cost as it has to traverse only one index, and this index is smaller since only the 1-dimensional distance values are used in the internal nodes. Figure 4.13b and Figure 4.13c shows the similar trends for color histogram. and wavelet feature datasets respectively.

Figure 4.14 provides the CPU cost of three indexing schemes for three datasets. From Figure 4.14a, we can see that as the dimensionality increases, the gap becomes wider between iLDR and gLDR. iMMDR is the best. Performance difference between iMMDR and iLDR is relatively small. When the dimensionality reaches 30, the CPU cost for gLDR is an order of magnitude higher than that for iMMDR and iLDR. The main reason is clear. In gLDR indexing structure, tree nodes contain multi-dimensional data points. However, in extended iDistance structure, tree nodes contain 1-dimensional key values. Extended iDistances (iMMDR and iLDR) incur single

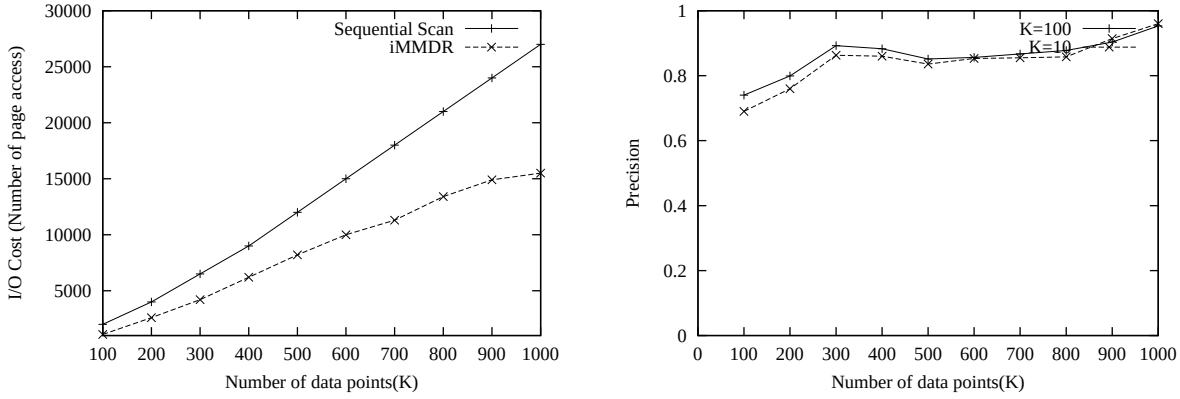
dimensional value comparison in searching while L-norm computation is involved in the Global structure's Hybrid-Tree. Thus computation in gLDR is much more expensive. Again, Figure 4.14b for color histogram dataset and Figure 4.14c for wavelet feature dataset shows the similar trend as Figure 4.14a. In terms of both CPU and I/O cost, the single dimensional extended iDistance index outperforms the Global indexing structure significantly. Furthermore, more effective dimensionality reduction method leads to more efficient indexing.

4.5.3 Scalability

All high-dimensional indexes are affected by the data size and number of dimensions. In this experiment, we look at scalability of MMDR. We set the data stream ratio ε as 0.005, and the k value in the lookup table to be 3 and the number of iterations that indicates a point as *inactive* as 10. The parameter we used here is the total response time (TRT) for MMDR to generate the optimal subspaces from the original data.

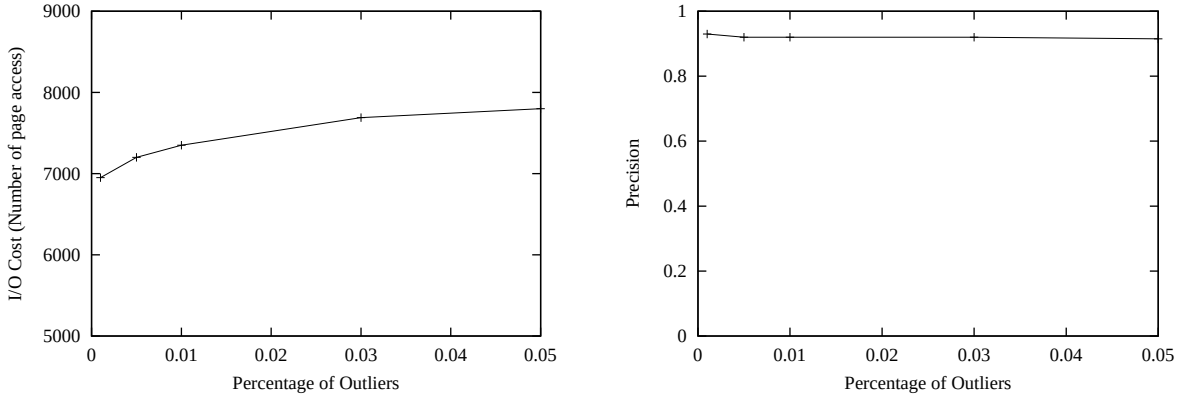
Figure 4.15a describes the effect of data size on the total response time. We keep the number of dimensions fixed at 100, while we vary the data size from 50,000 to 1,000,000. From Figure 4.15a we make the following observation. The response time increases linearly to the data size. When the data size reaches the limit of buffer - 500K, there is no jump in response time for scalable MMDR since we need only scan the whole dataset once. Figure 4.15b shows the effect of the number of dimensions on the total response time. For this experiment, we used 1,000,000 data points and varied the number of dimensions from 50 to 200. As expected, the total response time is nearly quadratic to the dimensionality. The results again exhibit that the limited buffer has no effect on the total response time.

4.5.4 Effect of Dynamic Insertions



(a) I/O cost (b) Precision

Figure 4.16: Effect on dynamic insertion



(a) I/O cost (b) Precision

Figure 4.17: Effect on outliers

The dimensionality reduction algorithm and the associated indexing structure must be dynamic and adaptive in order to handle dynamic insertions effectively. In this experiment, we see how Dynamic MMDR adapts and keeps its precision high, while keeping the KNN search cost low. For this experiment, we used the 1,000,000 100-dimensional dataset. We used 10NN and 100NN queries. We constructed the

index using the first 100,000 points and then we inserted 100,000 points at a time. At each insertion step, we performed KNN queries, and measured their query cost and precision.

Figure 4.16a shows that as more data points are inserted, a higher I/O cost is incurred. This is due to a bigger index, rather than the inefficiency of the index. However, the performance gain of iMMDR over sequential scan becomes larger and larger. Figure 4.16b shows that as more data points are inserted, the algorithm adapts dynamically and yields better precision. This again confirms the suitability of Mahalanobis distance in identifying effective clusters for dimensionality reduction.

4.5.5 Effect of Outliers

Outliers are data points that do not form effective clusters and are not included in any clusters obtained during dimensionality reduction. They form a cluster of its own, and they are indexed in their actual data space. Since outliers compose of 'abnormal' points, its cluster's maximum radius is very large after mapping them into 1-dimensional distance values. Hence it is searched for most queries. We used the 1,000,000 100-dimensional dataset. Figure 4.17a shows that as more outliers are introduced, the higher I/O cost is incurred. However, as shown in Figure 4.17b, it is interesting to note that outliers do not affect query precision severely since they are grouped into outliers set and are searched like other clusters.

4.6 Summary

In this chapter, we have presented an effective and fast dimensionality reduction algorithm – Multi-level Mahalanobis-based Dimensionality Reduction, which is able to

reduce the number of dimensions while keeping the precision high, and able to effectively handle large datasets and dynamic insertions. We used an extended iDistance to index the data points in different reduced subspaces. We conducted extensive experimental studies using both real and synthetic datasets to compare the algorithm with existing approaches. The results show that the proposed technique, as a whole, is very effective and efficient in supporting KNN search in very high-dimensional space. Furthermore, it is scalable for very large databases and able to hand dynamic insertions adaptively.

Chapter 5

Indexing Hyper-dimensional Image Feature

5.1 Introduction

An image feature, like texture, can contain hundreds or even thousands of dimensions. Such hyper-dimensional databases pose significant problems to existing high-dimensional indexing techniques which have been developed for indexing databases with (commonly) less than a hundred dimensions.

The problem of indexing and searching in a hyper-dimensional database is a challenging one, due to three main reasons:

- First, according to several studies (e.g., [9]), the expected minimal distance between any two points in a hyper-dimensional space is very large (becoming larger with increasing dimensionality) while the difference between the minimal and maximal distance to a point is expected to be small (becoming smaller with increasing dimensionality). These two characteristics of a hyper-dimensional space mean that the search radius for a k-nearest neighbor query is expected to be large. This in turn results in a large number of "false positives" since most points are expected to have almost equal distance to the query point. This

phenomenon leads to significant deterioration of the query performance in most existing indexing methods.

- Second, due to the extremely high dimensionality, the fanout for most indexes built on a hyper-dimensional space is typically very small, resulting in an increase in the height of the indexes (e.g., in a 200 dimensional space, we can't expect more that ten entries in an 8K page if 4 bytes are needed for each dimension).

- Finally, the computation of the distance (e.g., Euclidean distance) between two points in a hyper-dimensional space, becomes processor intensive as the dimensionality increases. This implies that the processor time is expected to become a significant portion of the overall query response time for a hyper-dimensional database. Proposed techniques for optimizing the performance of most indexing techniques do not take this into consideration.

In this chapter, we propose an effective methodology called Local Digital Coding (LDC) for finding KNN in a hyper-dimensional space. LDC is developed to address the problems mentioned above and provide a substantial reduction on both I/O and processor time when searching on hyper-dimensional datasets consisting of hundreds of dimensions. It is compatible with ubiquitous indices, such as B^+-trees and thus can be easily deployed.

Given a cluster of points in a high-dimensional data space, LDC transforms each point into a bitmap which we refer to as the point's Digital Code (DC). Each dimension of the point is represented by a single bit in its DC. The DC of a point is generated by comparing the coordinates of the point with the coordinates of the cluster center the point belongs to. A bit is set to 1, if the value of the dimension it corresponds to, is larger than the value of the corresponding dimension of the cluster

center, and 0 otherwise.

Since there is a bit in the DC for each dimension, indexing a D-dimensional space will result in DCs with D bits. The data points in a cluster can thus be separated into 2^D partitions with points in each partition sharing the same DC. Based on LDC, we propose a novel searching algorithm, called **Searching on-the-fly by PArtial-distance** (SPA). Given the DCs of both the query point and a partition, SPA **dynamically** selects a **subset** from the DCs (say n bits) to perform matching. A partition is pruned off if the number of matching bits in the two DCs is less than m bits. The intuition behind such an approach is that the points in the pruned partition are on different sides of some cutting planes with respect to the query point and thus are too far away to be in the answer set. A summary of our contributions is as follows:

- We present LDC, an efficient methodology for indexing a hyper-dimensional space. LDC has the following advantages over existing methods for handling hyper-dimensional data:
  - LDC is optimized to take both processor and I/O time into account. This is important since processor time for comparing vectors becomes significant in a hyper-dimensional space. Our work is the first to address this important problem to the best of our knowledge.
  - LDC's method of spatial inference for pruning the search space is based on cutting-planes instead of direct distance computation. Because of this, the proposed algorithm, SPA, that performs KNN search, has significantly less false "positives" compared to methods using direct distance computation.
  - LDC adopts a bit representation instead of storing the actual dimensional values, thus, its storage requirements are the smallest possible (without considering the option of using compression methods) for representing the

location of any D-dimensional partition. This effectively reduces the number of I/Os that are required to search the index. While LDC exploits bit representation, it is fundamentally different from the VA-file in three ways: 1) LDC uses one bit for each dimension, while the VA-file uses multiple bits. 2) Pruning in LDC is achieved dynamically by selecting a subset of DC for bitwise operation, while the VA-file involves full distance computation between approximation vectors. 3) LDC is adaptive to skewed data distributions while the VA-file has been shown to be effective mainly for uniform datasets [45].

- We identify the parameters that affect the performance of LDC and present the results of a detailed analysis identifying how these parameters can be tuned for certain objectives given a query point.

- We present the results of a detailed experimental evaluation of our methodology supporting our claims that LDC is capable of handling hyper-dimensional databases more efficiently, compared to other applicable methods.

The rest of this chapter is organized as follows. LDC is presented in Section 5.2. In Section 5.3, the novel SPA algorithm is introduced. An extensive performance study on our method is presented in Section 5.4. Finally, Section 5.5 summarizes this chapter.

5.2 Local Digital Coding (LDC)

In this section, we start by presenting the LDC methodology. Since LDC can easily co-exist with B^+-tree indices, we refer to a B^+-tree employing LDC as an *LDC tree*. Then, we will discuss how to efficiently construct an LDC tree.

5.2.1 Structure of the LDC tree

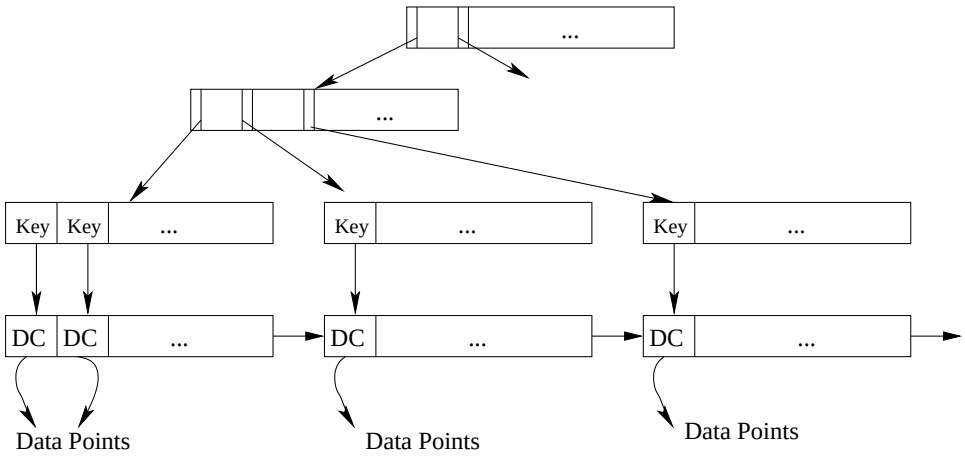


Figure 5.1: The overall structure of an LDC tree.

The structure of the LDC tree can be essentially separated into three levels. Figure 5.1 presents an example. The first level of the LDC tree is a B^+-tree which is formed by simply transforming each point in the hyper-dimensional space into a one-dimensional value. Several methods can be utilized to perform this transformation; we choose the method proposed in [62], due to its simplicity. In [62], the whole data space is first grouped into clusters with each point in the data space belonging to one of the clusters.

Assuming each cluster is assigned a unique ID, the one-dimensional value for a data point P in cluster i is computed as:

$$key(P) = i * c + d(P, O_i)$$

where O_i is the center of cluster i, c a constant and $d(.)$ a distance function, assessing the distance between two points in the hyper- dimensional space. The minimum and maximum value of $Key()$ among all points in a cluster, define a range of values, which we refer to as the *range* of cluster i. Constant c, which we refer to as the *scaling constant* is chosen as to increase the range of each cluster, so that the ranges

of different clusters are disjoint and can thus be distinguished easily. This way, the range of each cluster i, is $[i * c, (i + 1) * c]$. An additional auxiliary array is used to store the center and range for each cluster. Given a D-dimensional point P, the cluster with center O, that P belongs to, can be determined by computing $Key(P)$ and checking membership of the value $Key(P)$ in a cluster's range.

The second level of the LDC tree is the Digital Code (DC) level. The basic idea of a DC is to produce an effective single bit representation for each dimension of a data point. Given a D-dimensional data space, we encode each data point into a D-dimensional DC representation by applying the algorithm in Figure 5.2. Given a D-dimensional point P and a cluster center O, initially, the DC is set to 0 (line 1). For each dimension, if its value is equal or greater than the value of the corresponding dimension of the cluster center, we set the corresponding bit of the DC representing that dimension to 1 and 0 otherwise (lines 3 and 4).

The third level of the tree is the data level where the actual data points are stored.

Local Digital Coding Algorithm

```
LDC(P,O)
Input: P, O;
Output: DC;
1.  unsigned int DC=0;
2.  for i=0 to D-1
3.          if (P[i] ≥ O[i])
4.                  DC | = 1 << i;
5.  return DC;
```

Figure 5.2: Local Digital Coding Algorithm

5.2.2 Constructing the LDC Tree

The LDC tree is built in three steps. In the first step, the data space is partitioned into clusters. A plethora of proposals exist for clustering high dimensional data sets

[24, 63]. For hyper-dimensional datasets, this process can be computationally expensive even as a pre-processing step. A variety of approaches exist for clustering high dimensional datasets approximately, trading accuracy for speed [10]; such techniques can be readily utilized in our setting, significantly improving the performance of this preprocessing step.

For simplicity we adopt the following approach for partitioning the data set into clusters. We first select the edge points as done in [62] and then group the data points into clusters by assigning them to the nearest edge point. While doing so, the centroid of each cluster can be approximated. This is done by estimating the median of the cluster on each dimension through the construction of a histogram along each dimension. The centroid of each cluster is then used as the cluster center.

The $Key()$ value for each point (relative to its cluster center) is then computed. A single B^+-tree index is then built on these values. The second step links each distance value in the leaf nodes of the B^+-tree to the corresponding point's DC representation in order. This way the DC level of LDC is populated. Pages containing DC's are sequentially stored so that a sequential scan on DCs can be performed.

Finally, the third step links each DC to the corresponding data point in order. This way, for each point in the data space its corresponding DC value is under its corresponding $Key()$ value in the leaf nodes of the B^+-tree, and above the actual coordinates of the point.

Dynamic maintenance operations arising from insertions/deletions on an LDC Tree can be easily performed. When a new point is added in the dataset, its cluster membership is first determined by identifying the nearest cluster center. Then its one-dimensional distance and DC representation is computed, followed by the standard insertion operation in B^+-trees. Correspondingly, the point's linked DC and data are inserted into the correct position in the DC level and data level of the tree respectively.

Notation	Description
Q, P, O	Query, data point, cluster center
$Q[i]$, $P[i]$, $O[i]$	The i^{th} dimension value for Q,P,O
DC_Q, DC_P	The DC for Q and P
$DC_Q[i]$, $DC_P[i]$	The corresponding bit for the i^{th} dimension in the DC of Q and P
$rank$	A dimension ranking array
$rank[i]$	The i^{th} element of the dimension ranking array
D	Dimensionality of the database
DIM	The set of dimensions for the database, $\|DIM\| = D$
DIM'	A proper subset of DIM
s, t	Number of dimensions
m	Pruning length
n	Candidate pruning length
pd	Partial distance
Θ	Threshold for search space or $\frac{n}{m}$
Φ	Maximal number of candidates allowed

Table 5.1: A Table of Notations in Chapter 5

Deletion operations are quite similar. After the point's one-dimensional distance value is computed, the standard deletion operation in a B^+-tree is then performed, followed by removing the corresponding linked DC and data point.

5.3 KNN Query Processing

We now turn our attention on hyper-dimensional searching and show how a K-Nearest Neighbor (KNN) search can be effectively performed on the LDC tree. Without loss of generality and to simplify our presentation we will assume that the distance function adopted in the search is the Euclidean distance. Thus, in the remainder of this chapter $d(.)$ refers to the Euclidean distance between a pair of points. We summarize the notation used in Table 5.1 for quick reference. We first present the theoretical foundation of our algorithm.

5.3.1 Partial Distance

We begin by formally defining *partial distance*:

Definition 5.3.1 *(Partial Distance $pd(Q, P, DIM')$).* Let Q and P be two points in a D-dimensional space and let $Q[i]$, $P[i]$ denote the values of dimension i in Q and P respectively. We denote the set of D dimensions as DIM. Given $DIM' \subset DIM$, the partial distance between Q and P is defined as

$$pd(Q, P, DIM') = \sqrt{\sum_{i \in DIM'} (Q[i] - P[i])^2}$$

□

Thus, the partial distance between two D-dimensional points P and Q is in fact the Euclidean distance computed on a subset of the D dimensions. In what follows, when the parameters of partial distance are clearly implied by the context, we will denote partial distance simply using pd. We thus have the following corollary:

Corollary 5.3.1. Let $d(Q, P)$ denote the Euclidean distance between Q and P, then $pd(Q, P, DIM') \leq d(Q, P)$. □

Next, we derive a lower bound on the distance of a query point Q and a data point P, from their corresponding DC representations:

Theorem 5.3.1. Let Q be a query point and P be a data point. We use DC_Q and DC_P to denote the DCs that are computed with respect to a cluster center O. In addition, we use $DC_Q[i]$ and $DC_P[i]$ to denote the corresponding bit in dimension i in both DCs. Let DIM' be a set of dimensions such that $\forall i \in DIM', DC_Q[i] \neq DC_P[i]$, then

$$d(P, Q) \geq \sqrt{\sum_{i \in DIM'} (Q[i] - O[i])^2}$$

Proof:

Since DC_Q and DC_P differ on the i^{th} bit, this implies that Q and P are located on different sides of a plane that is parallel to the i^th dimension and passes through O. We can thus infer that $|Q[i] - O[i]| \leq |Q[i] - P[i]|$ for all $i \in DIM'$. Thus, we conclude that

$$\sqrt{\sum_{i \in DIM'} (Q[i] - O[i])^2} \leq pd(Q, P, DIM')$$

Corollary 5.3.1 completes the proof.

□

We now provide an **upper bound on the number of mismatches** that can occur between DC_Q and DC_P (computed with respect to O) if the distance between Q and P is less than a pruning distance - pd. To do so, we first introduce the notion of a **dimension ranking array**.

Definition 5.3.2 *(Dimension Ranking Array).* Let Q be a query point and O be a cluster center. The **dimension ranking array** for Q and O is an array, $rank$, of D positions such that:

1. Each dimension in the data space is uniquely represented by one element $rank[i]$ in the array.

2. $(Q[rank[0]] - O[rank[0]]) \geq Q[rank[1]] - O[rank[1]]) \geq ..... \geq Q[rank[D-1]] - O[rank[D-1]])$

□

Thus, each element $rank[i]$ in the dimension ranking array for O and Q is the dimension in which $|Q[rank[i]] - O[rank[i]]|$ is the i^{th} largest among all the D dimensions.

Theorem 5.3.2. Let DC_Q and DC_P be the DCs of two D-dimensional points which are computed with respect to a cluster center O. Let $rank$ be the dimension ranking array for Q and O. Given a pruning distance pd, we use s to denote the largest number such that

$$\sqrt{\sum_{i=D-s}^{D-1} (Q[rank[i]] - O[rank[i]])^2} \leq pd$$

If $d(P,Q) \leq pd$, then the number of bit mismatches between DC_Q and DC_P cannot be more than s.

Proof:

Assume DC_Q and DC_P have t mismatches, $t \geq s$ and let DIM' represent the set of dimensions in which the t mismatches between DC_Q and DC_P occurred. According to Theorem 5.3.1,

$$d(P,Q) \geq \sqrt{\sum_{i \in DIM'} (Q[i] - O[i])^2}$$

To minimize the R.H.S of the inequality, we should pick DIM' to be the set of t dimensions in which $|Q[i] - O[i]|$ are the smallest i.e. the dimensions represented by $rank[D-t]$ to $rank[D-1]$. Since s is the largest value such that

$$\sqrt{\sum_{i=D-s}^{D-1} (Q[rank[i]] - O[rank[i]])^2} \leq pd$$

and $t > s$, we can thus conclude that

$$\sqrt{\sum_{i=D-t}^{D-1} (Q[rank[i]] - O[rank[i]])^2} > pd$$

By combining the inequalities, we will have $d(P,Q) > pd$. □

Figure 5.3 illustrates the relationship between D, s and t on the ranking array. Notice that given a pruning distance pd, Theorem 5.3.2 enables pruning of candidate

data points. Given a query Q, a cluster center O and a pruning distance pd, one can derive the suitable value of s according to Theorem 4.2 and all DCs, with more than s mismatches to the query DC, can be pruned away. Such a strategy for pruning has two main drawbacks: (a) it involves performing bitwise operations on vectors of D bits, for a possibly very large D and (b) it provides no way to control the amount of actual pruning performed.

To overcome these drawbacks, we will compare the DCs on *only those bits that correspond to the top (largest) n dimensions* in the dimension ranking array. Comparing on only $n < D$ dimensions effectively reduces the number of bits examined, offering additional savings in processor time. Assuming n is sufficiently large, we can still determine a value m based on Theorem 5.3.2, such that having more than m mismatches among the n bits will mean that $d(P, Q) > pd$. Comparing on the n largest dimensions, for a specific value of pd, assures that n is as small as possible. Moreover, as we will see, it enables a flexible strategy that allows the user to control the amount of pruning performed.

We note that selecting a good value for n can be difficult. A large value of n will not bring substantial savings in processor time. Even more importantly, a small value of n could result in a low probability of pruning off candidate data points. For example, in the extreme case where $n = 1$ and $m = 0$, only 1 bit of the DCs is compared and there is only a 50% chance of pruning off a candidate DC if we assume a uniform distribution of the data points. In the next section, we will illustrate the relationship between the probability of pruning a partition and the selection of n and m.

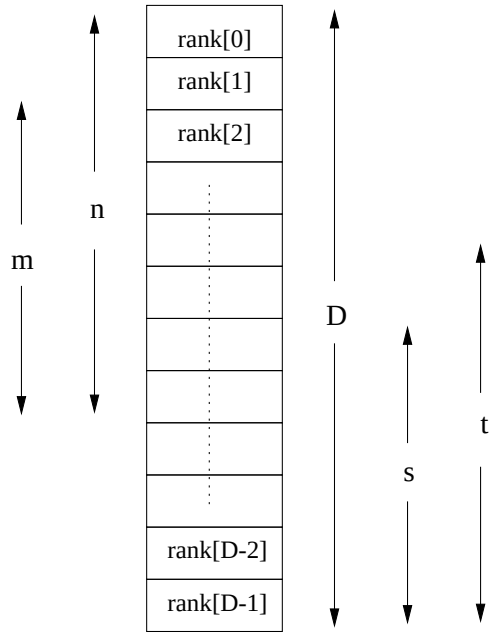


Figure 5.3: Dimensions Ranking Array.

5.3.2 Selecting the values m and n

In our analysis on the relationship between n and m, we shall assume that data are uniformly distributed in the hyper-dimensional space.

Lemma 5.3.1. A D-dimensional DC representation divides the portion of the hyper-dimensional space spanned by the points belonging to a cluster into 2^D partitions.

Proof: Since the i^{th} bit of a DC represents the two sides of a plane that is orthogonal to the i^{th} dimension and passes through a cluster center O, having D bits indicates that there are D orthogonal planes passing through O. Each of these mutually independent planes divides the original data space into half which gives us 2^D partitions. □

Figure 5.4 shows an example in 2-dimensional space. It is evident that the cluster space is divided into four partitions by two orthogonal planes. The partitions are

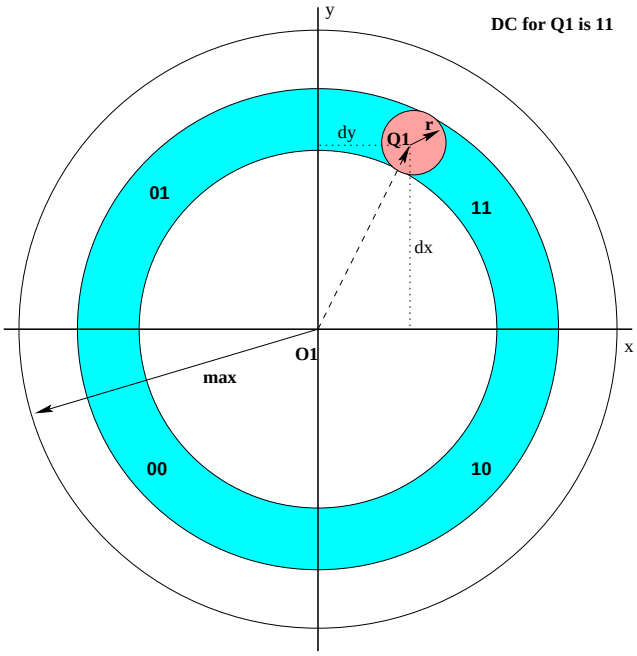


Figure 5.4: Searching space in a 2-d space

identified by their DCs as 11, 10, 01, and 00 respectively.

Lemma 5.3.2. Let DC_Q be a query DC and DIM' be a subset of any n dimensions. If comparisons are made only on the n bits of the DCs that correspond to DIM', then the average number of partitions which share exactly m common bits with DC_Q is

$$\begin{pmatrix} n \\ m \end{pmatrix} * 2^{(D-n)}$$

Proof: We first notice that each axis-aligned hyperplane, in expectation, will partition the points in the cluster into half. Hence, each bit in a data point's DC has equal probability of matching or not matching the corresponding bit of DC_Q. The probability that a DC has exactly m common bits with DC_Q is thus $\begin{pmatrix} n \\ m \end{pmatrix} * (\frac{1}{2})^m * (\frac{1}{2})^{n-m} = \begin{pmatrix} n \\ m \end{pmatrix} * \frac{1}{2^n}$. Since we consider only n bits, there will be 2^n different combinations

of the n bits. The number of DCs associated with each of these combination will be $2^{(D-n)}$. Thus, the average number of DCs with exactly m common bits with the query DC is $\begin{pmatrix} n \\ m \end{pmatrix} * \frac{1}{2^n} * 2^n * 2^{(D-n)} = \begin{pmatrix} n \\ m \end{pmatrix} * 2^{(D-n)}$. □

From Lemma 5.3.2, the have the following corollary:

Corollary 5.3.2. When comparing with a query DC on a subset of n bits, the average number of DCs which have no more than m mismatches with the query DC is:

$$\sum_{k=n-m}^{n} \begin{pmatrix} n \\ k \end{pmatrix} * 2^{(D-n)}$$

or equivalently

$$\sum_{k=0}^{m} \begin{pmatrix} n \\ k \end{pmatrix} * 2^{(D-n)}$$

The next theorem provides an estimate on the average percentage of partitions that are accessed when pruning is performed based on a subset of n bits from the query DC.

Theorem 5.3.3. If a bitmap of length n, extracted from the query DC, is compared with the partitions of a cluster and partitions are pruned if there are more than m mismatches between the query and partition DCs, then the percentage of partitions that are searched is:

$$Prob(n, m) = \frac{\sum_{k=0}^{m} \begin{pmatrix} n \\ k \end{pmatrix}}{2^n}$$

Proof: From Corollary 5.3.2, the number of partition DCs having no more than m mismatches with the query DC has an expected value of $\sum_{k=0}^{m} \begin{pmatrix} n \\ k \end{pmatrix} * 2^{(D-n)}$. Since

the total number of partitions is 2^D, the percentage of partitions being searched will be: $Prob(n,m) = \sum_{k=0}^{m} \binom{n}{k} * 2^{(D-n)}/2^D = \frac{\sum_{k=0}^{m} \binom{n}{k}}{2^n}$. □

The expression for $Prob(n,m)$ indicates that increasing n or reducing m results in a reduction of the search space. Hence by adjusting the values of n and m, we can control the number of candidates. We refer to such a value of m as the *pruning length* and n as the *candidate pruning length*.

The ideal case in a KNN search is to maximize pd and minimize $Prob(n,m)$. When pd is larger, the KNN can be discovered earlier, since a larger pruning distance will return more neighbors (hopefully K). However, when $Prob(n,m)$ is too small (a small m and a large n will result in a very small partial distance), a small fraction of a cluster is searched and thus, too few candidates are returned. In this case, there is a danger to filter out the KNN and further search has to be performed. So there is a trade-off between $Prob(n,m)$ and pd.

From Theorem 5.3.2, given specific values for n and pd, we derive that m is the *largest* number such that

$$\sqrt{\sum_{i=n-m}^{n-1} (Q[rank[i]] - O[rank[i]])^2} \leq pd$$

The above formula indicates the relationship among n, m and pd. Given m and pd, the n will be the *largest* number (to minimize $Prob(n,m)$) which satisfies the following *partial distance constraint*:

$$\sqrt{\sum_{i=n-m}^{n-1} (Q[rank[i]] - O[rank[i]])^2} \geq pd$$

We refer to this formula as the *partial distance constraint*. Theorems 5.3.2, 5.3.3, and the *partial distance constraint* pave the way for the search by partial distance algorithm - SPA.

5.3.3 The KNN Search Algorithm

Having established the relationship among m, n, and the partial distance, we are now ready to present the KNN search algorithm.

KNN Algorithm in LDC
Input: Q - Query point
Output: knn[] - k-nearest neighbors to Q
Variables:
    O[N]: Array of N cluster centers
    rank[][]: Rank array for each cluster center
    QDC[][] : Query DC for each cluster center
    Δr : Step for adjusting search radius r
    $\Delta\Theta$: Step for adjusting search space
1. for i=0 to N-1
2. rank[i] $\leftarrow$ rank_dimension(Q,O[i]);
3. QDC[i] $\leftarrow$ LDC(Q,O[i]);
4. set r, Θ, Φ;
5. do //start searching
6. candidates[] $\leftarrow$ SPA(Q,QDC[],rank[],r,O[],Θ,pre_r,pre_pd[]);
7. if candidates.size $>$ Φ
8. Θ = Θ - $\Delta\Theta$;
9. continue;
10. if candidates.size $<$ K
11. Θ = Θ + $\Delta\Theta$;
12. continue;
13. knn[] $\leftarrow$ compute_distance(candidates[],Q);
14. r = r + Δr;
15. until the K^{th}-NN is found

Figure 5.5: Main KNN Search Algorithm in LDC

Figure 5.5 shows the main routine for the KNN search in the LDC tree, given the query point Q. At the start of the search, the dimension ranking array is first filled based on relative position of the query point to the cluster centers in O (line 2). Relative to each cluster center, a DC for the query point can be computed based on the LDC algorithm in Figure 5.2 (line 3).

The search process is iterative in nature. It consists of incrementally adjusting the

search radius (by Δr), until all the KNNs are discovered (line 5-15). The incremental search process is controlled by three parameters, r the *search radius*, Θ, the desired *maximal fraction of space to search* and Φ, the desired *maximal number of candidates*. The initial values of these parameters are set in line 4.

Conventionally, we determine the initial value of r by the difference between the index key of the query point and index key of its nearest point in the same leaf node, where the query point possibly resides. The parameter, Θ, indicates the desired maximal fraction of space to be searched within the whole data space. It will be adjusted to control the number of candidates being considered as searching process goes on. Its initial value can be estimated based on sampling or from past historical records. While such an approach will work for uniformly distributed data, things are less straightforward if the data distribution varies in different part of the data space. In such situation, the number of candidates cannot be accurately estimated based on Θ. Hence having the same Θ value may correspond to largely different number of candidates for different queries. To ensure retrieval efficiency, small numbers of candidates are always desired. Φ, the desired maximum number of candidates, is used to adjust Θ to avoid overflow by candidates. We defer the discussion on Δr and Φ to Section 5.3.5.

The SPA method (line 6), utilizes partial distances to rapidly prune the candidates to a very small set by searching on-the-fly, employing $Prob(n, m)$. In particular, the threshold Θ and the search radius r are used to compute the suitable pair of (n,m) such that the number of candidates being considered is less than the desired fraction Θ of the points in the database. The parameter *pre_r* is used to detect if the search radius has been increased from previous SPA call. And the parameter *pre_pd*[] remembers the previous pruning distances to avoid the same computation as previous SPA call did. Both are initialized to 0 before the search starts.

In the KNN algorithm, the parameter, Θ can be adjusted (line 7-12) if the number of candidates returned by SPA is larger than Φ. By reducing Θ, fewer candidates will be produced in the next round of search (line 8). On the other hand, if the number of candidates is smaller than K, Θ will be increased to include more candidates (line 11). $\Delta\Theta$ can be set based on the volume of candidates returned from SPA. Higher volume corresponds to a larger $\Delta\Theta$, and vice versa.

If the K^{th}-NN's distance is equal to or smaller than the $min(pre\_pd[i])$, all KNNs are found (line 15). Notice that the minimal value among all $pre\_pd[i]$, instead of r, is the final pruning distance.

Now we proceed to the core of the search algorithm - the Search on-the-fly by PArtial-distance (SPA) algorithm. SPA consists of three steps: allocate the DCs, generate suitable values of m and n, and prune data by DC comparison. Figure 5.6 describes the algorithm.

Given a search radius r and a query Q, SPA first checks if the searching space intersects with a cluster range. For each cluster center i, O[i].min and O[i].max indicate the min and max values in the cluster range. If they intersect, SPA allocates two key values nearest to the values of $max(i * c - dist - r, i * c + O[i].min)$ and $min(i * c + dist + r, i * c + O[i].max)$ respectively in the cluster space (line 6-7), followed by allocating their corresponding DCs (line 8-9). Since all DCs at the DC level of the LDC tree are linked to each other, DCs within a range can be sequentially scanned and retrieved for DC comparison thus avoiding random I/O access.

Next, SPA aims to identify the pair (n, m) such that $Prob(n, m)$ satisfies the desired fraction of space to search - Θ (line 10-16). Given a pd value which is initialized to be r, the method $generate\_pruning\_length$ tests values of m from 1 to D and generate the corresponding n by enforcing the *partial distance constraint*. The (n, m) pair which minimized $Prob(n, m)$ can be computed this way. If the minimum $Prob(n, m)$

SPA Algorithm

```
Input: Q, QDC[], rank[], r, O[], Θ, pre_r, pre_pd[];
Output: candidates[];
1. for i=0 to N-1
2.    dist = compute_distance(O[i], Q);
3.    if O[i].min > dist+r and O[i].max+r < dist
4.        continue;
5.    // allocate the starting and ending DCs
6.    l_leaf ← alloc_leaf(btree,max(i*c-dist-r,i*c+O[i].min));
7.    r_leaf ← alloc_leaf(btree,min( i*c+dist+r,i*c+O[i].max));
8.    l_DC ← alloc_DC(l_leaf);
9.    r_DC ← alloc_DC(r_leaf);
10.   // generate m and n given a pd as pruning distance
11.   pd = r;
12.   do
13.       generate_pruning_length(n,m,pd,Q,rank[i],O[i]);
14.       pd = pd - Δr;
15.   while Prob(n,m) ≥ Θ & (r=pre_r || pd > pre_pd[i])
16.   pre_r = r; pre_pd[i] = pd + Δr;
17.   // pruning on-the-fly
18.   for each DC from l_DC to r_DC
19.       num_diff_bit=0;
20.       for j=0 to n-1
21.           if DC[rank[i][j]] & 1 ≠ QDC[i][rank[i][j]] & 1
22.               num_diff_bit++;
23.       if num_diff_bit < m
24.           add DC to candidates[];
25. return candidates[];
```

Figure 5.6: SPA Algorithm

is still too large, pd will adjusted to a smaller value so that a new (n, m) pair (and corresponding a new minimum $Prob(n, m)$) is generated.

After finalizing the pair (n, m), further processing takes place by application of Theorem 5.3.2 (line 17-24). Any DC with m or more different bits from the query DC is safely pruned. At this step, the DC comparison involves only bitwise operations on n bits (instead of D) making it CPU efficient.

Consider Figure 5.4 again. A cluster space is divided into four partitions by the cluster center. Given a search radius r, pd is first initialized to r. Since $pd < d_y$ and $pd < d_x$, the best (n, m) pair is identified as (2,1). Thus, any point in the space having a DC with 1 or more mismatches can be pruned. The query DC is encoded as 11, thus the partitions identified by DCs of 10, 01, and 00 can be safely pruned. The final searching space is the light shaded area in the partition identified with a DC of 11. We shall use another example to illustrate the concept of SPA.

Example 5.3.1. *An Example on SPA:* For illustration purposes, we assume the L_1 norm (instead of L_2) as the metric function. Consider a $2NN$ query with query point Q=(0.9, 0.1, 0.55, 0.7, 0.35) and a cluster center O=(0.5, 0.6, 0.5, 0.5, 0.5). The respective DC, the indexing key and rank array for Q are depicted in Table 5.2, and a cluster of data points is shown in Table 5.3. It can be derived manually that the 2NNs are P_3 and P_5 with distance 0.4 and 0.42 respectively.

Our main KNN algorithm initializes the search radius $r = 1.0$, $\Theta = 0.4$ and $\Phi = 3$. We set Δr to 0.5. SPA first allocates the left node by a point search on the B^+-tree on key value = max(dist(Q,O)-r,O.min)=1.05. Similarly, the right node is allocated by point search on key value = min(dist(Q,O)+r,O.max)=2.3. Obviously, all the DCs, except P_9's, will be scanned for DC comparison.

Next, SPA generates a pair (n, m) which minimizes $Prob(n, m)$, given the value

Q	$(0.9, 0.1, 0.55, 0.7, 0.35)$
$\|Q[i] - O[i]\|$	(0.4,0.5,0.05,0.2,0.15)
key	1.3
DC	10110
$rank$	[1,0,3,4,2]

Table 5.2: A query with its key, DC and rank.

of pd. When pd=1.0 and under the partial distance constraint of

$$\sum_{i=n-m}^{n-1} (|Q[rank[i]] - O[rank[i]]|) \geq pd$$

the pair (5,4) will minimize $Prob(n, m)$. However, since $Prob(n, m)$ is still higher than Θ, the value of pd will be reduced by Δr to have a value of 0.5. With this new value of pd, SPA will minimize $Prob(n, m)$ to a value of $\frac{5}{2^4}$=0.31 by setting (n, m) to (4,2).

In the final step, since (n, m) is set to (4,2), the candidates with 2 or more different bits in the 4 most distinguishable dimensions (from the rank array, we can see that they are dimension 1,0,3 and 4) will be pruned. In this example, only P_3 and P_5 will remain. Since the number of candidates is less than Φ and equal to K, only data points P_3 and P_5 are accessed.

After computing the distance on the full dimension, the distance to the second nearest neighbor is 0.42, which is less than pd=0.5 (notice that current r is 1.0), and the search stops.

5.3.4 Optimizing the Generation of (n, m)

There are three main cost terms comprising the overall cost of SPA. First, it is the point search in the LDC tree. Since a point search in a B^+-tree is extremely fast, this part can almost be neglected. Second, it is the cost of generating the pair (n, m). And third, it is the cost of comparing DCs using bitwise operations.

P	$Data$	key	DC
P_1	(0.1,0.9,0.3,0.55,0.0)	1.45	01*0*10
P_2	(0.35,0.2,0.95,0.8,0.9)	1.7	00*1*11
$\mathbf{P}_3$	(0.85,0.15,0.6,0.65,0.45)	1.1	10*1*10
P_4	(0.2,0.8,0.65,0.95,0.4)	1.2	01*1*10
$\mathbf{P}_5$	(0.92,0.15,0.4,0.6,0.25)	1.32	10*0*10
P_6	(0.65,0.8,0.1,0.4,0.3)	1.05	11*0*00
P_7	(0.15,0.9,0.3,0.1,0.7)	1.45	01*0*01
P_8	(0.4,0.1,0.25,0.7,0.75)	1.3	00*0*11
P_9	(1.0,0,0.99,0.05,0.95)	2.49	11*0*11

Table 5.3: A cluster of data points with keys and DCs.

It is evident that for large values of n and m, computing $Prob(n, m)$, becomes prohibitively expensive. The following theorem alleviates this complication:

Theorem 5.3.4. As the value of $\frac{n}{m}$ increases, $Prob(n, m)$ decreases if n>2m.

Theorem 5.3.4 states that as $\frac{n}{m} \uparrow$ (increases), $Prob(n, m) \downarrow$ (decreases). Hence, the problem of identifying the (n, m) pair that minimizes $Prob(n, m)$ becomes equivalent to the problem of identifying the (n, m) pair with the maximal $\frac{n}{m}$ value. This observation leads to a much more efficient procedure for setting the parameters (n,m). The meaning of the Θ threshold has to be refined however, as it becomes a *threshold for the value* $\frac{n}{m}$. By doing so, the computational cost reduces to computing n given m and the partial distance constraint. Furthermore, the range of m values to be tested (identifying n given a pd value, such that $\frac{n}{m} > \Theta$) can be less than $\frac{D}{\Theta}$ and greater than the value of m, such that (assuming the L_1 distance is used) $\sum_{i=0}^{m-1} (Q[rank[i]] - O[rank[i]]) > pd$.

Let us revisit Example 5.3.1. We use $\frac{n}{m}$ rather than $Prob(n, m)$ and set Θ to be 2. Given pd=1.0, we find that m should be larger than 3. Since the dimensionality is 5, m should be less than $\frac{5}{2}$. Thus no suitable m can be produced. In the next iteration, a value $pd = 0.5$ is used. In this iteration, we conclude that m should be larger than 1. Thus, only m=2 is tested and n=4 is generated. Since $\frac{4}{2}$ =2 which is equal to Θ,

the pruning length and candidate pruning length are generated. From this example, we can see that $\frac{n}{m}$ and $Prob(n, m)$ have the same capability to control the number of candidates. However, by using $\frac{n}{m}$, the computational cost is very little. In the rest of paper, Θ refers to the threshold value of $\frac{n}{m}$.

5.3.5 A Cost Model

The I/O cost of SPA can be calculated as:

$$IO_{total} = IO_{B^+} + IO_{DC} + IO_{candidate} \approx IO_{DC} + IO_{candidate}$$

where IO_{B^+} is the number of pages required for point search on a B^+-tree, IO_{DC} is the number of pages required for sequential scan at the DC level, and $IO_{candidate}$ is the number of pages required to access the candidate data points. IO_{B^+} is very small and can be neglected, as B^+-trees have guaranteed logarithmic access cost.

An index technique has to be more efficient than sequential scan to be effective. A sequential I/O is usually a factor of 10 times faster then random access. Hence one of the important factors affecting the I/O cost is the number of candidates for random access. Next, we derive the proper value for Φ (the maximal number of candidates allowed for random access to ensure the efficiency of the indexing structure) for LDC tree. In the LDC tree, the size of DC level is only $\frac{1}{32}$ (assuming 32-bit precision) of data level. The KNN in the LDC tree is iterative and parts of DC level are scanned in each iteration. In the worst case, the whole DC level is scanned (only when the whole space is searched). Notice that each candidate is accessed only once. To ensure SPA is more efficient than sequential scan, the following relationship has to be true assuming that a sequential I/O is 10 times faster than random I/O:

$$\Phi * 10 + \alpha * \frac{Page_{scan}}{32} \leq Page_{scan}$$

where α represents the number of iterations for a query to finish its KNN search and $Page_{scan}$ represents the total number of pages for the whole data set. Following the above formula, we have:

$$\Phi \leq \frac{(32-\alpha)}{320} * Page_{scan}$$

From the above formula, we can see that in the best case when only one iteration is processed, Φ can reach nearly 10% of the data size. Generally speaking, running tens of iterations for KNN is not desirable. Setting Δr based on the difference between the cluster's minimal and maximal radii can prevent the number of iterations to be large. Even when α reaches 10 or more, let's say 16, Φ can still be as large as 5% of the data size. The above formula assumes that the whole DC level is scanned in each iteration. In a real situation, only part of the DC level is scanned. Hence Φ can be set to even a larger value.

The processor cost of SPA can be calculated as:

$$CPU_{total} = CPU_{n,m} + CPU_{DC} + CPU_{candidate} \approx CPU_{DC} + CPU_{candidate}$$

where $CPU_{n,m}$ is the computational cost to generate the (n, m) pair, CPU_{DC} is the cost to compare DCs by performing bit operations, and $CPU_{candidate}$ is the cost to compute the distance on the complete set of dimensions on the candidate data points. Using the optimization of Section 5.3.4, the cost of $CPU_{n,m}$ becomes negligible.

5.4 Performance Study

Otherwise stated, all experiments were performed on a Sun UtraSparc II 450Mhz (2 CPU), with 4G memory. We used a page size of 4K (different page size may have different effect on the results. However it is not our focus here). Unless otherwise

stated, all results reported are averages over 100 queries, for 10NN. We adopted the L_2 norm for indexing and searching. Our experiments were conducted using the following data sets:

1. **Real datasets**: We created one data set extracted from 73,715 WWW images randomly crawled from over 40,000 web sites by ICICLE system. It consists of 159-dimensional color histograms extracted from these images.

2. **Synthetic datasets**: We have created several uniformly distributed datasets in order to be able to effectively vary the dimensionality. The first four sets of 100K points each are in 128-, 216-, 512-, and 1024-dimensional space. We used three additional datasets consisting of 50K, 100K, and 200K 1024-dimensional data points to evaluate the effect of increasing data size on SPA. Note that a 1024-dimensional data point spans the 4K page boundary and hence a random access of each point involves reading two pages. The last two sets we used, consist of 100,000, 512-dimensional data points constructed by generating each coordinate independently using a Zipfian distribution, with skew parameters 0.7 and 1.0 respectively.

The most popular techniques in the literature, suitable for indexing spaces of high dimensionality are the VA-file[59] and the IQ-tree[6]. Each dimension in the VA-file is uniformly represented by 5 bits. We also tested the effectiveness of iDistance [61]. Our experiments indicated that the IQ-tree structure does not scale gracefully to very large dimensionality. Figure 5.7 presents the results of an experiment showing SPA's, VA-file's, IQ-tree's and iDistance's total response time as the dimensionality of the underlying uniform data space increases [1]. Among all methods, SPA performs best.

[1]This experiment was conducted under windows XP with a Pentium III processor (600Mhz) and 128RAM.

It is evident that the total response time for IQ-tree increases nearly exponentially with respect to the dimensionality. This phenomenon is pronounced when the dimensionality reaches hundreds or higher. The total response time for the IQ-tree is two times that of sequential scan in a 256-dimensional space. The main reason for IQ-tree's behavior, in our opinion, is the use of Minimum Bounding Rectangles (MBRs). In very high dimensionality, there are three main drawbacks associated with MBRs: First, the distance between each MBR and random query points becomes extremely small. Most MBRs intersect with the searching spheres defined by queries, which leads to accessing most of the points in the underlying space. Second, the distance computation between an MBR and a query is expensive in a data space of very high dimensionality. Third, the representation of each MBR is too large. Given a D-dimensional space, representing MBRs in this space requires 2D-dimensional vectors. For a 512-dimensional space given a page size of 4K, the size of each MBR is exactly the page size (assuming 4 byte coordinates).

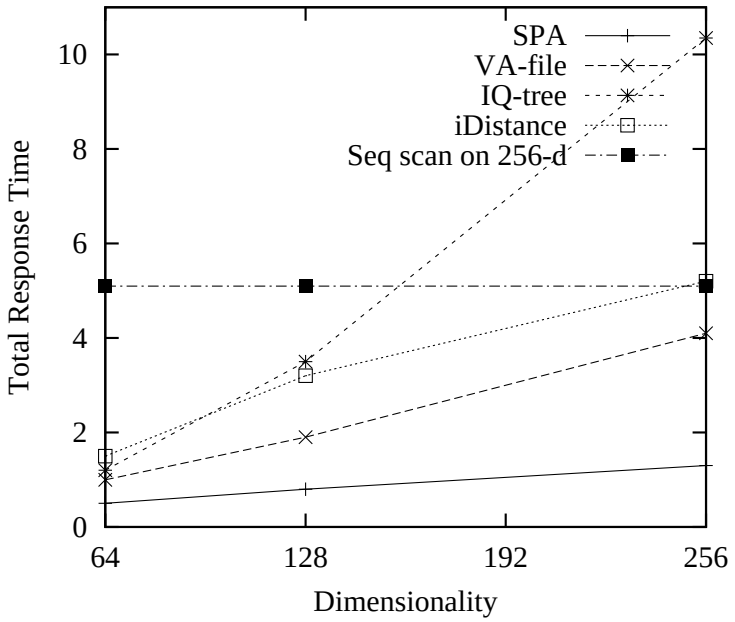


Figure 5.7: Effect of dimensionality on total response time.

For iDistance, its performance degrades with the increase in dimensionality and is slightly worse than sequential scan in a 256-dimensional space. The reason is obvious. As the dimensionality increases, the transformation from full dimensionality to one dimensional distance value causes more information loss. Thus the ability to prune the search space is degraded. Clearly, more candidates will be included for random access as the dimensionality increases.

The performance of the VA-file structure, which is better than the IQ tree and iDistance, is known to be linear to the dimensionality and better than sequential scan if the dataset is uniformly distributed. Thus, we adopt the VA-file as a basis for our comparison.

5.4.1 Effect of Θ

In SPA, the Θ (or the threshold value for $\frac{n}{m}$) affects the number of candidate accesses at the data level of the LDC tree. When the dimensionality is up to hundreds, the search radius r needs to be set very large which eventually requires a sequential scan of a large portion of the DC level of the LDC tree.

SPA can automatically drive the ratio $\frac{n}{m}$ to approach Θ by adjusting the value of *pd*. In the following experiment we investigate the effect that different (n, m) pairs have on the total number of disk accesses, given a specific value of *pd* qualified to return the correct 10NN. When the (n, m) pair, maximizes the ratio $\frac{n}{m}$, the best performance is achieved. Here we use an 100K uniform dataset of dimensionality 128. Figure 5.8 shows the effect of $\frac{n}{m}$ on page IO. As expected, when $\frac{n}{m} \uparrow$, the I/O cost drops. When $\frac{n}{m}$ reaches its maximum value, the I/O cost is minimal. Clearly, the larger the Θ is, the less the I/O cost becomes.

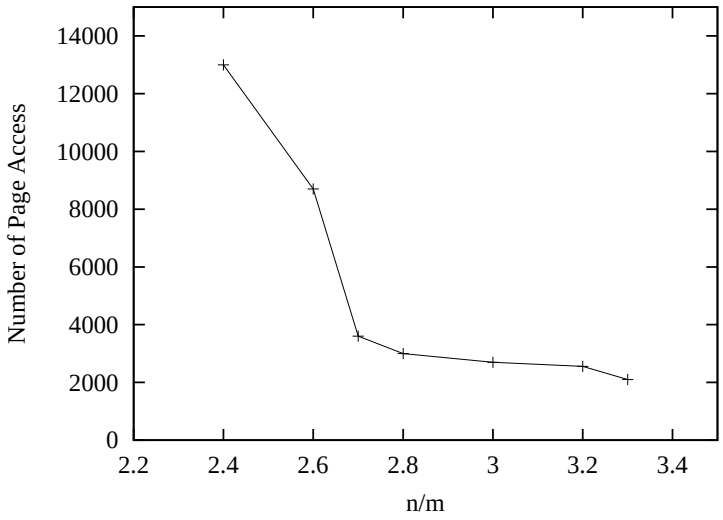


Figure 5.8: Effect of $\frac{n}{m}$ on I/O.

5.4.2 Effect of Φ

One interesting thing about SPA is that it can control the number of candidates to be less than Φ by adjusting Θ. Hence it is necessary to know the relationship between the retrieval precision for KNN search and the number of candidates. Next, we investigate such a relationship in SPA by testing uniform datasets with 100,000 points and the real dataset. Figure 5.9 shows the precision for uniform datasets as the number of candidates increases for different dimensionalities. We can see that the precision is not sensitive to the dimensionality. For all dimensionalities, the precision reaches nearly 100% after the number of candidates is above 3000, which is around 3% of the data size. Hence 3% of data size might be a good choice for the value of Φ for uniform datasets. Figure 5.10 shows the precision for the real dataset as the number of candidates increases. As we can see, the precision grows up to almost 100% after the number of candidates is more than 400, which is less than 0.6% of the data size. Comparing with 3% of the data size for uniform datasets, the color histogram dataset requires a much smaller number of candidates relatively to the data size. This is due

to the skewness of the color histogram dataset.

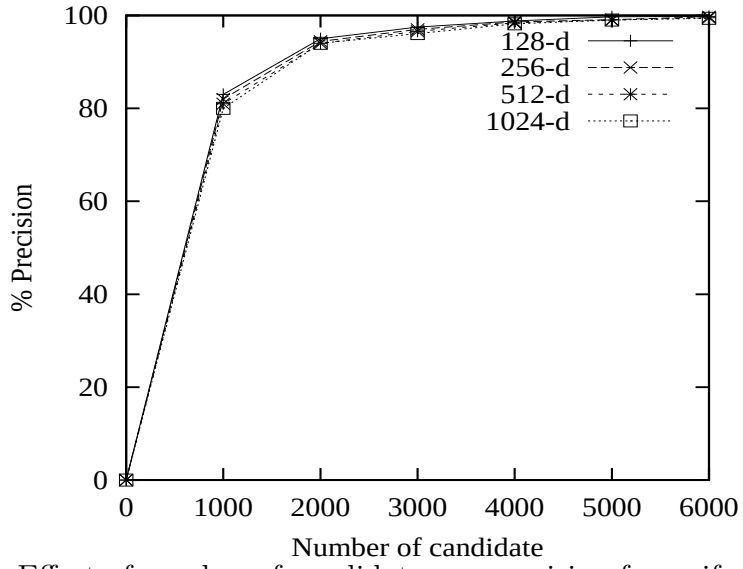


Figure 5.9: Effect of number of candidates on precision for uniform datasets.

5.4.3 Effect of Data Size

We now investigate the effect of data size on both I/O and processor cost. In this experiment, we tested a 1024-dimensional uniform dataset varying its size from 50K to 200K, and an 159-dimensional color histogram, varying its size from 10K to 70K.

Figure 5.11 shows the improvement achieved by SPA over the VA-file. Both the I/O and processor cost of VA-file increase faster than that of SPA, as the data size increases. It is evident that SPA offers an improvement over the IO and processor cost of VA-file, by nearly four times. Sequential scan on processor cost is not presented as its cost is *too large to fit* in the figure. This is also applied to the following figures.

The I/O cost of SPA mainly consists of two parts: sequential scan on part of the DCs and access to the candidate data points. The I/O cost of the VA-file consists of sequential scan on the VA-file and access to the candidate data points. The DC level of the LDC tree is just $\frac{1}{5}$ of the size of VA-file. Although the KNN search in

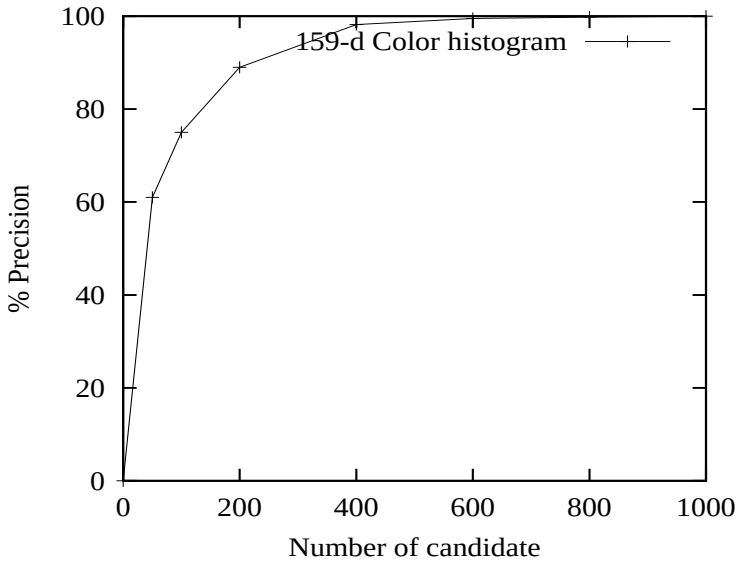
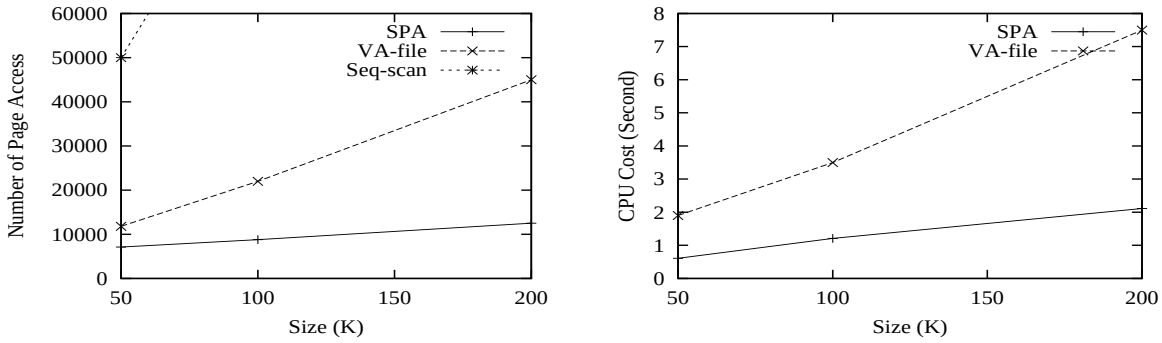


Figure 5.10: Effect of number of candidates on precision for real dataset.

the LDC tree is iterative (i.e., DC level may be scanned more than one time), it can outperform the VA-file since SPA controls the number of candidates.

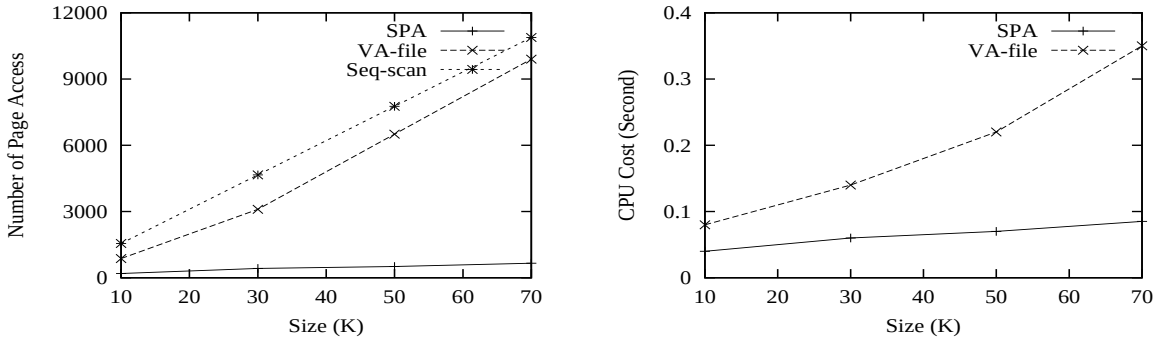
The main processor cost for SPA is the DC comparison, where only bit operations are involved. However, in the VA-file, there is a distance computation for every VA. In a very high dimensional space, the full dimensional distance computation is too expensive. Figure 5.11b presents an experiment that verifies this statement. In this figure the processor cost for the VA-file reaches 7 seconds.

Figure 5.12 presents the results on the 159-dimensional color histogram dataset. In terms of the total I/O cost (Figure 5.12a), the VA-file performs close to a sequential scan. In contrast, SPA only requires a small number of I/Os. In this experiment, SPA improves the I/O cost over the VA-file by more than an order of magnitude. Notice that the color histogram is a skewed dataset (few dimensions have large values, many have 0 or close to 0 values). Comparing with Figure 5.11a which presents the results of the same experiment on a uniform dataset, we observe that SPA performs even better in skewed datasets while the performance of VA-file is much worse. Recall that



(a) Effect on I/O (b) Effect on CPU

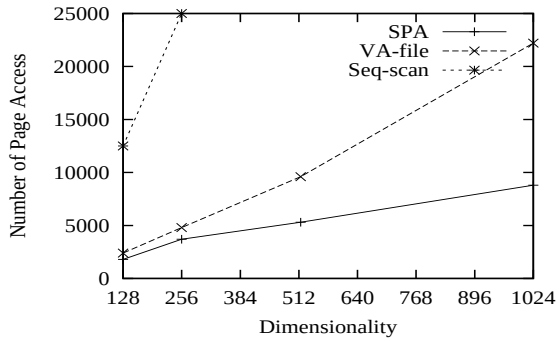
Figure 5.11: Effect of Data Size on Uniform Dataset.



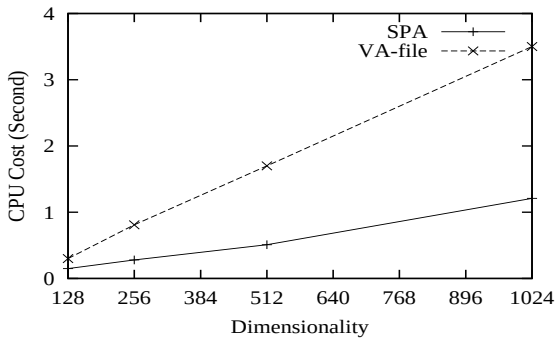
(a) Effect on I/O (b) Effect on CPU

Figure 5.12: Effect of Data Size on Color Histogram Dataset.

SPA ranks the dimensions with respect to local reference points (cluster centers) and chooses the most distinguishable dimensions as the candidate pruning dimensions so that a larger partial distance can be used. In color histograms, the most distinguishable dimensions are obviously those dimensions with large values. Hence, comparing to uniform data, by keeping the ratio $\frac{n}{m}$ large, a smaller value of candidate pruning length is used and a smaller number of candidate data points remain for full distance comparison, In contrast to SPA, the VA-file includes most of the data points as candidates when it computes its lower and upper bounds for skewed data. Figure 5.12b again shows the significant advantages in terms of processor time of SPA over the VA-file.

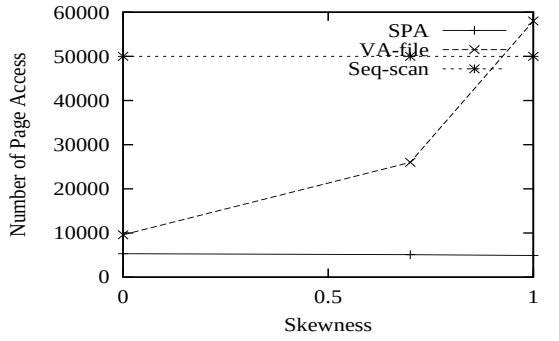


(a) Effect on I/O

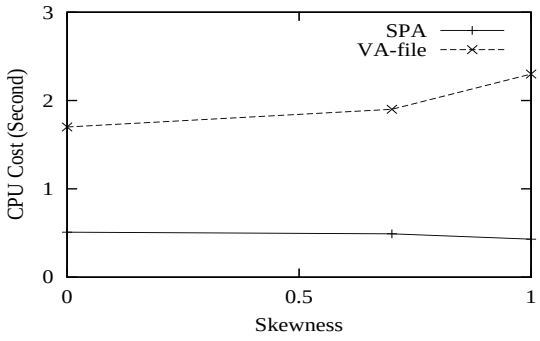


(b) Effect on CPU

Figure 5.13: Effect of Dimensionality on Uniform Dataset.



(a) Effect on I/O



(b) Effect on CPU

Figure 5.14: Effect of Data Skewness.

Our experiments indicate that SPA attains superior performance on skewed data sets, as shown by our experiments with image color histograms, in Figure 5.12.

5.4.4 Effect of Dimensionality

With this set of experiments we demonstrate the robustness of SPA as the dimensionality of the underlying data space increases. We use uniform datasets of size 100K points, of dimensionality 128, 256, 512, and 1024 respectively. Figure 5.13a demonstrates that SPA outperforms the VA-file more substantially as the dimensionality increases. The higher the dimensionality, the better is the performance of SPA. This

is mainly because the number of candidate points tested in SPA is independent of the dimensionality and the cost of accessing DCs increases much slower than in the case of the VA-file. Figure 5.13b again confirms the performance advantages of SPA over a VA-file on processor time.

5.4.5 Effect of Skewness

In this experiment, we will demonstrate that SPA is even more effective for skewed datasets, where the KNN search is more meaningful. In skewed datasets, the most skewed dimensions are expected to be better amenable to pruning, since SPA chooses the most distinguishable (or skewed) dimensions to maximize the partial distance. Thus, comparing with uniform datasets, SPA is expected to have better performance.

We generated two 512-dimensional data sets, by populating the coordinates using a Zipfian distribution with skew parameter 0.7 and 1.0 respectively. Both consist of 100,000 data points. Figure 5.14 shows the effect of skew on both SPA and the VA-file. Figure 5.14a indicates that as the skew parameter of the Zipfian distribution increases, the I/O cost of VA-file increases rapidly. When the skew parameter becomes 1.0, the performance of VA-file is even worse than that of sequential scan. In contrast, SPA still performs much better. The performance gap between the two methods gets rapidly much wider. In real life datasets that are mostly skewed to a certain degree, we expect SPA to achieve excellent performance as demonstrated in Figure 5.12 using real data sets. Similar observations can be made for processor costs as shown in Figure 5.14b.

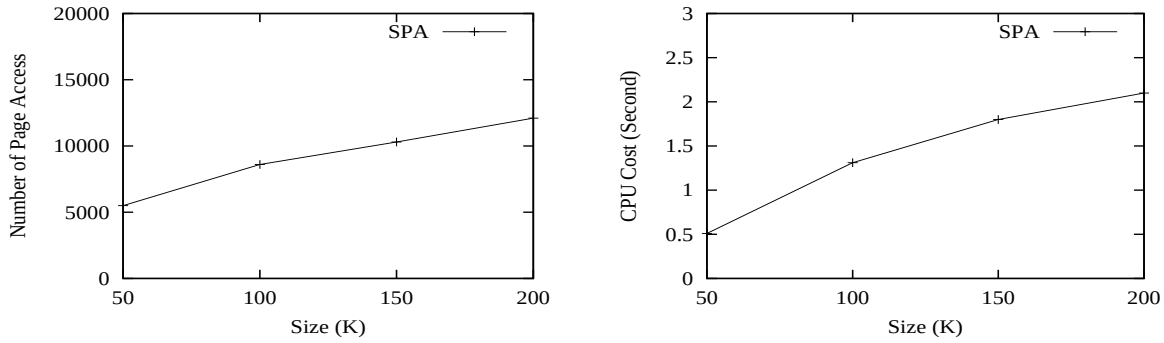
5.4.6 Effect of Dynamic Insertion

In this experiment, we test the effect of dynamic insertion on the efficiency of our method. We used the 1024-dimensional uniform dataset. We constructed the LDC tree using the first 50,000 points and then we inserted 50,000 points at a time. After each insertion, we performed KNN search and computed their average costs.

Figure 5.15 depicts the changes of I/O cost and CPU cost as more data points are inserted. As we can see, higher I/O costs and CPU costs are incurred. Obviously, a larger index incurs more cost. However, we notice that the increasing rate of cost becomes lower and lower than that of data size as more and more data points are inserted. Comparing with Figure 5.11 which presents the results of the index rebuilt upon each bulk insertion. we observe that dynamic insertion without rebuilding the index does not deteriorate the performance of SPA significantly. This confirms the effectiveness of our LDC tree structure. However, recall that the DCs of points are encoded based on their local cluster centers. As more points are inserted into clusters, the positions of cluster centers may change. Naturally, the movement of cluster centers will affect the effectiveness of DCs. When the movement is beyond certain threshold, there is a need to rebuild the index to achieve satisfactory performance.

5.4.7 Effect of LDC in Extended ICICLE

In this experiment, we show the effect of LDC on color-based retrieval in the extended ICICLE. The dataset we searched is the 159-dimensional color histogram extracted from 73,715 WWW image in ICICLE. We use the total response time by sequential scan as the baseline for comparison. Table 5.4 indicates the ratio of total response time spent by LDC and VA-file over sequential scan. As we can see, the total response time of VA-file is almost same as that of sequential scan (0.91 of that of sequential



(a) Effect on I/O

(b) Effect on CPU

Figure 5.15: Effect of Dynamic Insertion on Uniform Dataset.

Indexing Methods	Ratio
VA-file	0.91
LDC	0.06

Table 5.4: Ratio of total response time over sequential scan.

scan). However, the total response time of LDC is just 0.06 of that of sequential scan. Clearly, the LDC speeds up the searching by nearly 20 times. The integration into ICICLE and experiment illustrate the practicability of the LDC in a real setup. Indeed, the LDC could be applied effectively into a commercial DBMS.

5.5 Summary

In this chapter, we introduced a very effective data organization and representation methodology called Local Digital Code suitable for hyper-dimensional image features. Such representation encompasses the application of partial distance and accommodates a novel KNN search algorithm - SPA.

SPA uses the minimal partial distance computed from any m dimensions among n most informative dimensions between the query and static reference points (cluster centers), as the partial distance. Such partial distance computation avoids accessing

data points so that the overall computational costs are minimized. SPA is capable of pruning points in the data space rapidly, without computing distances among them, employing DCs and simple bitwise operations. Moreover, SPA can minimize the candidate point set that requires retrieval and further processing, by employing the results of our analytical methodology. Our extensive performance study on hyper-dimensional data demonstrated that SPA outperforms known methods significantly. Its improvement was also reflected from the performance of extended ICICLE.

Chapter 6

Indexing Multiple Image Features

6.1 Introduction

Images are usually described by multiple features, each of which is typically high-dimensional. In the context of CBIR, an image may be described by a 64-dimensional color histogram, a 16-dimensional texture histogram, a 2-dimensional size, and a 1-dimensional date.

To support multi-feature queries, we can build a high dimensional index on the feature space obtained from all dimensions of the multiple features. In our image example, this corresponds to an 83-dimension feature space. Unfortunately, such an approach is not likely to be effective because of the high dimension. Moreover, existing high-dimensional indexing techniques typically treat all the different dimensions homogenously. An alternative approach is to build one index for each feature. In this case, multi-feature queries are evaluated by integrating results from each index to get the final rank-ordered results. However, combining answers from multiple indexes for ranked queries may require examining a large portion of each index.

In this chapter, we devise a novel representation that compactly captures f multi-dimensional features into two vector components. The first component is an f-dimensional vector obtained by transforming each of the f features into a value in

a single dimension space. The second component is a bit sequence of size $2\sum_{i=1}^{f} d_i$ where d_i is the number of dimensions of the ith feature, i.e., each dimension contributes two bits. The bits are set by analyzing each feature's distance histogram. This representation leads to a two-level index structure where the first tier indexes the first component using a standard multi-dimensional index structure such as an R-tree, and the second level is a compact list of bit sequences accessible from the leaf node entries of the first level. Using our image example, we would create a top-level index on a four-dimensional feature vector, corresponding to the four features (color, texture, size, and date). Each of these features can then be further exploded into its full dimensionality in a second level of index - the bit sequence level.

Our technique results in more effective indexing, as we experimentally demonstrate, for several reasons. First, high-dimensional indexing is hard, and most systems attempt to reduce dimensionality to the extent possible. Our two level decomposition automatically brings about this dimensionality reduction (in our running example, the top level index has only four dimensions instead of 83). Second, explicit identification of semantically meaningful features makes it easy to weight these features as desired, on a per query basis. For example, a query that cares only about color and size (ignoring texture and date) as well as a query that cares about all four features can both be supported using one single index on image objects in our database. Third, high-dimensional indexing techniques often use a low-dimension projection for indexing [7, 62]. These techniques assume geometric homogeneity – all dimensions are considered equivalent – an assumption that is valid only within the dimensional attributes of a single feature. Our two-level decomposition permits these powerful reduction techniques to be applied one feature at a time.

We also propose a novel K Nearest Neighbors (KNN) query searching algorithm

that iteratively prunes the search space aggressively based on the most critical dimensions of highly selective features. The VA-file [59], which employs a bit representation of the feature vector, has been shown to be superior to sequential scan in high-dimensional space. We therefore evaluate our proposed structure against multiple-feature retrieval methods using single and multiple VA-files. Our extensive experiments on both real life and synthetic datasets indicate that the proposed index structure offers significant performance advantages over sequential scan and the VA-file based methods.

The rest of chapter is organized as follows. In in Section 6.2, we present the multi-feature representation model and indexing scheme. In Section 6.3, a novel KNN search scheme is presented, based on the multi-feature indexing scheme. An extensive performance study is reported in Section 6.4, and finally, we summarize this chapter in Section 6.5.

6.2 Representing and Indexing Multiple features

In this section, we first present our compact representation model for multiple features. We then present a two-tier index structure that can be used to facilitate searching under the model. Finally, we discuss how the basic model can be optimized.

6.2.1 A Compact Multi-Feature Representation

Let $P = (P^1, P^2, \ldots, P^f)$ be an f-feature point, where P^i is the ith feature and comprises d_i dimensions. We represent P as a two vector component (Dist, BitSequence) where Dist is a f-dimensional vector, and BitSequence is a bit sequence of length $2^{\sum_{i=1}^{f} d_i}$.

The Dist Component

The first component, Dist, is obtained as follows. We map each high-dimensional feature into a single-dimensional space. While several methods can be utilized to perform this mapping (e.g., the Pyramid technique [7] and iDistance[62]), we employ the following simpler transformation:

$$Dist^i(P, O) = L_P(P^i, O^i)$$

where P is a data point and O is the center of the data space, and $Dist^i$ is the distance value between P and O on feature i. L_p is the distance function applied.

Note that different features may have different dimensionality and may apply different distance functions. Before combining the distances from all features to generate the overall distance, each feature distance has to be properly normalized. We normalize $Dist^i$ into the range of [0,1] by the following Distance Normalization Formula (DNF):

$$Dist^i_n(P, O) = \frac{Dist^i(P, O) - Dist^i_{min}}{Dist^i_{max} - Dist^i_{min}}$$

where $Dist^i_{min}$ and $Dist^i_{max}$ are the minimal and maximal distance values in the ith feature space. $Dist^i_n$ is the normalized single-dimensional representation of feature i for point P. Thus, Dist of P is an f-dimensional vector given as follows:

$$Dist = (Dist^1_n, Dist^2_n, \ldots, Dist^f_n)$$

The BitSequence Component

The second component, BitSequence, is a bit sequence comprising $2\sum_{i=1}^{f} d_i$ bits. Each dimension of each feature contributes 2 bits based on the novel concept of a *distance histogram*.

Definition 6.2.1 *(distance histogram - dh).* Given a data point P and the data space center O, let P^i and O^i denote the ith feature of P and O respectively. We also denote the number of dimensions in the ith feature be d_i. Let $P^i[j]$ denote the value of the jth dimension of P^i. Then, $P^i_{min} = \min_{j=1}^{d_i} |P^i[j] - O^i[j]|$, and $P^i_{max} = \max_{j=1}^{d_i} |P^i[j] - O^i[j]|$. The distance histogram of P in the i^{th} feature, dh^i, represents the frequency distribution in which the widths of the bars are proportional to the distance intervals and the height of the bar is proportional to the number of dimensions whose distances (i.e., $|P^i[j] - O^i[j]|$) fall in the corresponding interval. □

Now we are ready to see the main routine, outlined in Figure 6.1, for generating the bit sequence of the ith feature of point P. Initially, the bit sequence for P^i is set to 0 (line 1)[1]. Then the dh^i of P^i is constructed (line 2). Based on the pattern of dh^i, we can determine a threshold value φ^i for the ith feature of P (line 3). We shall defer the discussion on φ^i to a later section. For each dimension, if its value is equal to or greater than the value of the corresponding dimension of the data center, we set the corresponding first bit of the sequence representing that dimension to 1 and 0 otherwise (lines 5 and 6). Similarly, if its distance to the corresponding dimension of the data center is equal to or greater than φ^i, we set the corresponding second bit of the sequence to 1 and 0 otherwise (lines 7 and 8). Geometrically, for each dimension, the first bit indicates to which side of the center it lies, and the second bit indicates if its distance to the corresponding dimension of the center is greater than φ^i.

The bit sequence generation algorithm essentially encodes the d_i-dimensional feature vector into a $2 \cdot d_i$ bit sequence, together with its corresponding distance threshold - φ^i. Thus, BitSequence of P is the concatenation (denoted $\oplus$) of all bit sequences generated for each feature using the above bit sequence generation algorithm:

$$BitSequence_P = B_P^1 \oplus B_P^2 \ldots \oplus B_P^f$$

[1]Multiple int or char can be used for different dimensionality.

Bit Sequence Generation Algorithm
BG(P^i, O^i)
Input: P^i, O^i;
Output: B_P^i;
1. unsigned int B_P^i=0;
2. $dh^i \leftarrow$ construct_dh(P^i, O^i);
3. $\varphi^i \leftarrow$ generate_φ(dh^i);
4. for $j = 0$ to d_i-1
5. if (P^i[j] $\geq$ O^i[j])
6. B_P^i | = 1 << 2*j;
7. if (|P^i[j] - O^i[j]| $\geq$ φ^i)
8. B_P^i | = 1 << 2*j+1;
9. return B_P^i;

Figure 6.1: Bit sequence generation algorithm.

To summarize, the proposed multi-feature representation is clearly compact. Instead of storing $\sum_{i=1}^{f} d_i$ dimension values for an f-feature point, we only need to require f distance values and 2 $\sum_{i=1}^{f} d_i$ bits with corresponding f threshold (φ) values. This clearly results in a significant reduction in storage when $d_i > 2$, which it almost always is. In the few cases where a simple feature has just one or two dimensions, we can store it as is, rather than use this bit encoding. All of our results still hold, and all algorithms we present below carry through unchanged except for additional condition checks to see for each feature whether it is BitSequence encoded. To minimize clutter, we ignore this straightforward special case for the remainder of this chapter, and focus our presentation on the more challenging high dimensionality case.

6.2.2 A Two-Tier Indexing Structure

The two vector component of the multi-feature representation lends itself very nicely to indexing. The first component can be used to prune away points based on the

distances within each feature: points that are similar will have similar distances, while those that are not will be far apart. Similarly, the second component zooms in to the dimension level of each feature. While the first component provides a global pruning, the second component serves as a more localized pruning.

Thus, we propose a two-tier structure where the first tier indexes the first component, and the second tier handles the second component. Moreover, since we do not expect the number of features to be very large (not more than 20), any existing multi-dimensional structures can be used for the first component. Since the second component is a compact bit sequence, storing and scanning a list of bit sequences is a practical solution. In our work, the first tier is an R-tree that is constructed by indexing on f dimensional vector obtained from normalized distance value of the points. The internal nodes of the R-tree are the standard Minimum Bounding Rectangles (MBR). Leaf nodes of the R-tree contain the f-dimensional transformed representations together with pointers to the second tier. The second tier is a list of bit sequences corresponding to the f-dimensional point. Each bit sequence also points to the data page containing the data point (in its full dimensions) that it represents. Figure 6.2 illustrates the proposed structure with an example comprising two features.

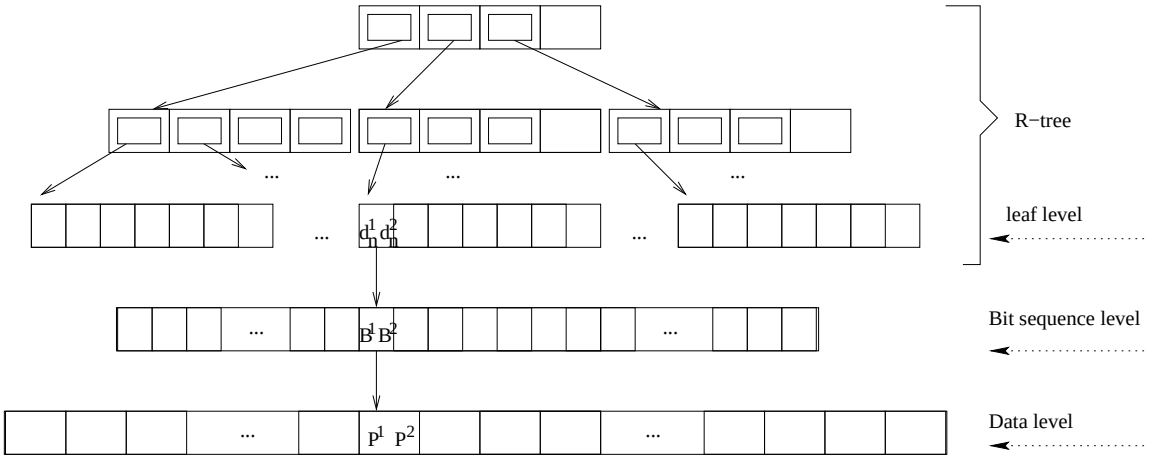


Figure 6.2: The indexing structure.

Dynamic maintenance operations arising from insertions/deletions on the two-tier structure can be easily performed. When a new point is added in the dataset, its Dist component and BitSequence component are first computed, followed by the standard insertion operation in R-trees by inserting Dist component into R-tree. Correspondingly, the point's BitSequence component and data are inserted into the correct position in the bit sequence level and data level of the structure respectively. Deletion operations are done in the similar way.

6.2.3 Tuning Bit Sequence Generation

Before leaving this section, we would like to discuss how the threshold value, φ^i for ith feature of a point, and the number of intervals (bars) for the distance histogram of ith feature, I^i, can be determined. Clearly, the distance histogram of a feature for different points will be different. As such, different φ^i values should be used for different points. Similarly, since different features may have different number of dimensions, the number of intervals for different features should be different also.

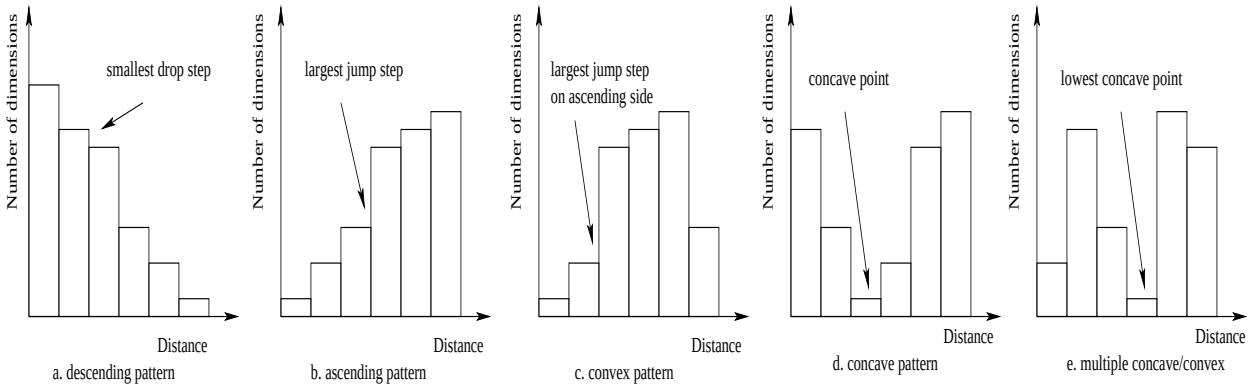


Figure 6.3: Patterns of distance histogram.

As a first cut, we adopt simple heuristics for determining φ^i here. This is based on the observation that there are mainly five types of trends in distance histograms, as shown in Figure 6.3. We analyze each of them here.

- Descending Pattern: This is the worst case since most dimensions have small distance to corresponding dimensions of the data center. In such circumstance, the value occurs on the *smallest drop step* is selected as φ^i value, as reducing the φ^i value by one more interval will include the smallest increase on the number of dimensions to be set to 1.

- Ascending Pattern: This is the best case since a large number of dimensions can be set to 1 with a large φ^i value. In this case, the value occurs on the *largest jump step* is selected as φ^i value, as reducing the φ^i value by one more interval will cause the largest decrease on the number of dimensions to be set to 1.

- Convex Pattern: In this case, most dimensions are centered at middle. To include interval with most dimensions, the value occurs on the *largest jump step on ascending side* of histogram is selected as φ^i value.

- Concave Pattern: In this case, most dimensions are distributed at both ends of histogram. The *concave point* (bottom interval) is a good choice for the φ^i value, excluding the bottom interval.

- Multiple Convex/Concave Pattern: Most histograms may have this pattern, especially when the number of intervals is big. In this case, we choose the *smallest concave point* as the φ^i value.

For I, we adopt the simple scheme that the number of intervals should be larger for features with larger number of dimensions. Thus, for the ith feature with d_i dimensions, we have its number of intervals computed as follows:

$$I^i = \lceil \frac{d_i}{c} \rceil$$

where c is a constant.

Notation	Description
pd, pd_{lb}	Partial distance, lower bounded pd
dh	Distance histogram
φ	Distance threshold value
I	Number of intervals
c	Constant for determining I
$Dist_n^i$	Normalized distance on feature i
P, P^i	A data point, feature i
B_P^i	Bit sequence of P^i
Q, O	A query point, Center of data space
$P^i[j]$	The j^{th} dimension value for P^i
d_i	Dimensionality of feature i
DIM^i	The set of dimensions for feature i
$DIM^{i'}$	A proper subset of DIM^i
RIC	Relative Information Carried

Table 6.1: A Table of Notations in Chapter 6.

6.3 KNN Query Processing

In this section, we shall present a novel KNN search algorithm. We note that range query is a special case KNN, and can be performed by our KNN algorithm with little modification. We shall thus not discuss it further. Before we look at the proposed algorithm, we will first introduce another concept, called *partial distance*, that allows us to lower bound the pruning distance between data points and queries in a multi-features environment for effective pruning by comparing their corresponding bit sequences. Table 6.1 summarizes the notation used in our discussion.

6.3.1 Lower Bounded Partial Distance

Definition 6.3.1 *(partial distance - pd).* Let Q and P denote a query and data point respectively. Let Q^i and P^i be the corresponding ith feature of both points. We denote the set of d_i dimensions as DIM^i. Given $DIM^{i'} \subset DIM^i$, the partial distance pd^i in

feature space i between Q and P is defined as

$$pd^i(Q, P, DIM^{i'}) = \sqrt[p]{\sum_{j \in DIM^{i'}} (|Q^i[j] - P^i[j]|)^p}$$

□

Thus, the partial distance between Q^i and P^i is the L_p distance computed on a subset of the d_i dimensions. We thus have the following corollary:

Corollary 6.3.1. $pd^i(Q, P, DIM^{i'}) \leq Dist^i(Q, P)$

□

Corollary 6.3.1 indicates that pd^i lower bounds $Dist^i$. Next, we define *lower bounded partial distance* from bit sequence representations.

Definition 6.3.2 *(lower bounded partial dist. - pd_{lb}).* Let B_Q, B_P, B_Q^i and B_P^i denote the bit sequences of Q, P, Q^i and P^i respectively. Let φ_P^i denote the φ of P^i. We also use $B_Q^i[2j]$ and $B_Q^i[2j+1]$ to denote the first and second bit for dimension j of B_Q^i. Let m=2j and $\epsilon = |Q^{\mathrm{i}}[j] - O^{\mathrm{i}}[j]|$, given $DIM^{i'} \subset DIM^i$, then for $\forall j \in DIM^{i'}$, we have:

$$pd_{lb}^i(\mathrm{Q}, B_P, DIM^{i'}) = \sqrt[\mathrm{p}]{\sum_{\mathrm{j} \in \mathrm{D}IM^{\mathrm{i}'}} \Phi[j]^p}$$

$$where\ \Phi[j] =$$

$$\begin{cases} \epsilon + \varphi_{\mathrm{P}}^{\mathrm{i}} \ \text{ if } \ \mathrm{B}_{\mathrm{Q}}^{\mathrm{i}}[m] \neq \mathrm{B}_{\mathrm{P}}^{\mathrm{i}}[m] \wedge \mathrm{B}_{\mathrm{P}}^{\mathrm{i}}[m+1] = 1 \\ \epsilon \ \text{ if } \ \mathrm{B}_{\mathrm{Q}}^{\mathrm{i}}[m] \neq \mathrm{B}_{\mathrm{P}}^{\mathrm{i}}[m] \wedge \mathrm{B}_{\mathrm{P}}^{\mathrm{i}}[m+1] = 0 \\ |\epsilon - \varphi_{\mathrm{P}}^{\mathrm{i}}| \ \text{ if } \ \mathrm{B}_{\mathrm{Q}}^{\mathrm{i}}[m] = B_{\mathrm{P}}^{\mathrm{i}}[m] \wedge \mathrm{B}_{\mathrm{Q}}^{\mathrm{i}}[m+1] \neq \mathrm{B}_{\mathrm{P}}^{\mathrm{i}}[m+1] \\ 0 \ \text{ if } \ otherwise \end{cases}$$

□

Notice that the input for pd_{lb}^i are Q, B_P, and $DIM^{i'}$ since we do not use data point P, but its bit sequence - B_P. From Definition 6.3.2, we have the following corollary:

Corollary 6.3.2. $pd^i_{lb}(Q, B_P, DIM^{i'}) \leq pd^i(Q, P, DIM^{i'})$

since

$$\forall i, \Phi[j] \leq |Q^i[j] - P^i[j]|$$

□

From corollaries 6.3.1 and 6.3.2, we can easily derive the following lemma:

Lemma 6.3.1. $pd^i_{lb}(Q, B_P, DIM^{i'}) \leq Dist^i(Q, P)$

□

Based on lemma 6.3.1, we next introduce our complex query searching algorithm - Adaptive Searching by Aggressive Partial-distance (ASAP).

6.3.2 Adaptive Searching by Aggressive Partial-distance

The conventional K Nearest Neighbors (KNN) searching process is iterative in nature. It initializes a starting radius and incrementally adjusts the search radius, until all the KNNs are found. We call a search radius on the entire data space as *complex r*, and on a single feature space as *atomic r*. Given a query Q, its distance to a point P is a combination of distances computed from all atomic queries. Without loss of generality, we assume that the importance of features are linearly combined. Thus the distance denoted as *distance*, between two data points P and Q with f features can be computed as follows:

$$Dist(P, Q) = \sum_{i=1}^{f} w^i * Dist^i_n(P, Q)$$

where $Dist^i_n$ is the normalized distance between P and Q on feature i, and w^i is a weight that represents the importance of feature i, and we have $\sum_{i=1}^{f} w^i = 1$.

ASAP essentially analyzes the pattern of the distance histogram of each feature of the query, and 1) *adaptively* selects the access order of the features, and 2) *aggressively*

ASAP Algorithm

1. $B_Q^{[]} \leftarrow$ GB(Q,O); //without setting second bit
2. $dh^{[]} \leftarrow$ compute_dh(Q,O);
3. $DIM^{[]}$' $\leftarrow$ select_DIM'($dh^{[]}$);
4. ao[] $\leftarrow$ select_accessing_order($dh^{[]}$, $DIM^{[]}$');
5. r= initial value;
6. do //start searching
7. r = r + Δr;
8. $r^{[]} \leftarrow$ get_atomic_r(r, $w^{[]}$);
9. left_right_nodes[] $\leftarrow$ search_Rtree(Q,$r^{[]}$);
10. B_P[] $\leftarrow$ scan_bitsequence_level(left_right_nodes[]);
11. for each B_P //compute $pd_{lb}^{[]}$
12. for n = 1 to f
13. $B_Q^{ao[n]} \leftarrow$ set_2nd_bit($\varphi_P^{ao[n]}$);
14. if $\sum_{i=1}^{n} w^i * \frac{pd_{lb}^{ao[i]} - d_{min}^i}{d_{max}^i - d_{min}^i} >$ r
15. break; // prune it
16. if n = f+1
17. add B_P's offset to candidate[];
18. $DIM^{ao[n]}$' $\leftarrow$ adjust_DIM'($DIM^{ao[n]}$');
19. knn[] $\leftarrow$ compute_knn(candidate[]);
20. until the K^{th}-NN is found

Figure 6.4: The ASAP KNN searching algorithm.

adjusts the partial distances based on increasingly larger subsets of dimensions for effective pruning. Figure 6.4 outlines the proposed ASAP KNN search algorithm. The algorithm mainly consists of the following steps:

Generate B_Q (line 1): Given a complex query Q, ASAP first generates its bit sequence representation without setting the second bit for each dimension. By comparing Q with respect to the space center O, the first bit is set, in each dimension. Since each point has a different φ for different feature, the second bit will only be set when the second tier of index is reached.

Generate dh (line 2): Next, the *dh* for each feature of query Q is constructed by comparing with O.

Select DIM' (line 3): At this step, a proper subset of DIM^i — $DIM^{i'}$ for each feature i of query Q is selected by analyzing its dh^i. The purpose of selecting $DIM^{i'}$ is for computing the lower bounded partial-distance at a later stage. Obviously, a larger number of dimensions in $DIM^{i'}$ will provide a larger pd^i_{lb}, but this will also lead to a higher computational overhead. Inspired by the dynamic φ generation scheme in Section 6.2.3, here we also dynamically select $DIM^{i'}$ based on the patterns of dh^i. The φ^i for feature i of query Q is first generated. Then we set the $DIM^{i'}$ to contain all the dimensions whose distances to the corresponding dimensions of O^i is greater than φ^i, i.e., any dimension j that satisfies the following condition will be included in $DIM^{i'}$:

$$|Q^i[j] - O^i[j]| \geq \varphi^i$$

Based on dh^i, $DIM^{i'}$ includes all dimensions lying in the right most intervals, starting from the φ^i (Figure 6.3). By doing so, $DIM^{i'}$ contains the most distinguishable dimensions of Q^i with respect to O^i. Meanwhile, the size of $DIM^{i'}$ is much smaller than the size of DIM^i. The advantage of such selection is that it is adaptive

to the pattern of distance histogram without tuning any parameter while achieving a satisfactory tradeoff between the size of $DIM^{i'}$ and the value of pd^i_{lb}.

Select Accessing Order (line 4): Given a multi-feature query Q, different features may have different capabilities in pruning search space. Accessing features in a decreasing order of pruning effectiveness will have fewer false 'positives' and so can avoid processing other features. This will significantly reduce the computational cost. To evaluate the pruning effectiveness of each feature of a query, we use the parameter called *Relative Information Carried* - RIC, which is defined as:

Definition 6.3.3 *(RIC).* Given a Q^i and its $DIM^{i'}$, the RIC for feature i is defined as

$$RIC^i = w^i * \frac{\sum_{j \in DIM^{i'}} |Q^i[j] - O^i[j]| - dist^i_{min}}{dist^i_{max} - dist^i_{min}}$$

□

In RIC, $DIM^{i'}$ is the set of dimensions selected in the last step since only the dimensions in $DIM^{i'}$ will be used. The total information carried by feature i is the sum of distances from Q^i to O^i for dimensions in $DIM^{i'}$. Since different features have different importance, the total information carried will be affected by respective importance. The features are then ordered based on their RIC values. Based on RICs, ASAP is able to *adaptively* manage the access order.

R-tree Pruning (line 7-9): The search starts with an initially incremented complex radius r. Recall that the R-tree is built on the f-dimensional distances transformed from f features. Since different queries may place different importance on a feature, the algorithm has to be able to handle various combinations of importance. To achieve this, r is first parsed to generate the atomic radius for each feature. Give a query Q, any point P qualified to be a candidate, we have:

$$\left.\begin{array}{l} w^1 Dist^1(P,Q) + ... + w^f Dist^f(P,Q) \le r \\ Min(Dist^i(P,Q)) = 0 \end{array}\right\} \Rightarrow Dist^i(P,Q) \le \frac{r}{w^i} \Rightarrow r^i = \frac{r}{w^i}$$

The above formula indicates that more important features will be searched with smaller radii in R-tree. Using derived atomic radii, a range rectangle search is performed in R-tree (line 9) and returns the left most and right most leaf nodes' offsets. Figure 6.5 shows an example with two features. Provided with r^1 and r^2 for Q^1 and Q^2 respectively, their single-dimensional searching spaces will be mapped into rectangles in R-tree. The results returned by R-tree are the intersection of their rectangles, as shown in Figure 6.4.

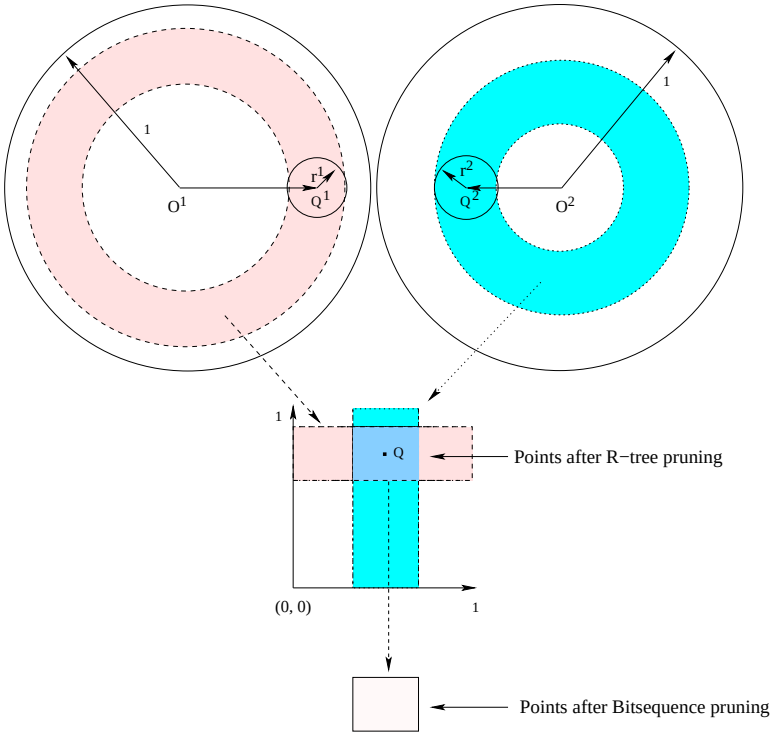


Figure 6.5: Space Pruning by ASAP.

Retrieve bit sequences (line 10): In the bit sequence tier, the starting and end nodes pointed from the left most and right most nodes in R-tree's leaf level are allocated and a sequential scan is then performed to retrieval all the bit sequence candidates.

Bit sequence pruning (line 11-17): At this step, we will see how pd_{lb} can be used to substantially reduce the number of candidate returned from R-tree search. For any B_P, the feature with the first order is first evaluated. The second bit of B_Q^i is then set by comparing to its corresponding φ_P^i (line 13). Then pd_{lb}^i(Q,$B_P, DIM^{i'}$) is computed based on Definition 6.3.2. If the accumulated partial-distance computed from the accessed features is greater than r, the point will be pruned (line 14). If the accumulated partial-distance for all features is not greater, it is a potential candidate. Hence the feature access order will affect the time of a point being pruned. Remember that $DIM^{i'}$ contains the dimensions with largest distance to the corresponding ones of data center. Hence we process the *most distinguishable features* first, and compute lower bounded partial-distance by the *most distinguishable dimensions* for each feature to reach the goal of pruning false 'positives' as soon as possible. As shown in Figure 6.5, the density of the final rectangle to be accessed after the bit sequence pruning step is greatly lower than that of the rectangle returned from the R-tree pruning step.

Adjust DIM' (line 18): Notice that after each iteration, as the search radius increases, more false 'positives' will be returned from the R-tree pruning step. To strengthen the pruning effectiveness, a larger pd_{lb} is needed to satisfy the pruning condition. To increase pd_{lb}, DIM' has to contain more dimensions. We aggressively adjust the DIM' by the following method. Recall that for a feature i (with its distance histogram dh^i and threshold φ^i), $DIM^{i'}$ contains all dimensions in the intervals from j to I^i where j is determined by φ^i. We then treat the intervals from 1 to $j-1$ as a new distance histogram and recursively apply the φ selection criterion on it to generate a smaller φ^i. We can then determine the intervals j' that corresponds to the new φ^i value. Dimensions in intervals from j' to $j-1$ are then added to $DIM^{i'}$. Hence DIM' will contain a larger number of most distinguishable dimensions. To

decide whether it is time to adjust, a threshold on the number of candidates can be used to control it. To avoid duplicate computation, pd_{lb} will be propagated to next iteration.

Compute KNN (line 19): In the final step, random accesses are performed on candidates and KNNs are computed. KNNs are found if the following strictest condition is satisfied:

$$Dist(knn[K-1], Q) \leq r$$

where knn[K-1] is the K^{th} nearest neighbor.

6.3.3 A Cost Model

Now we derive a brief cost model for ASAP in the two-tier structure.

The I/O cost of ASAP mainly consists of three parts: range searching in R-tree, sequential scanning in bit sequence level and random access of candidates in data level, i.e.,

$$I/O_{total} = I/O_R + I/O_{BS} + I/O_{Candidate}$$

where I/O_R, I/O_{BS} and $I/O_{Candidate}$ represent the I/O cost for R-tree searching, bit sequence scanning, and candidate access respectively.

Assuming R-tree is 50% full, the number of pages for R-tree can be approximately computed as:

$$Page_R \approx Page_{scan} * \frac{2f}{\sum_{i=1}^{f} d_i}$$

where $Page_{scan}$ is the number of pages for data.

Similarly, assuming each dimension is a 32-bit value, the number of pages for bit sequence level can be approximately computed as:

$$Page_{BS} \approx Page_{scan} * \frac{\frac{2}{32} * \sum_{i=1}^{f} d_i + f}{\sum_{i=1}^{f} d_i} = Page_{scan} * (\frac{1}{16} + \frac{f}{\sum_{i=1}^{f} d_i})$$

Since f is generally far smaller than $\sum_{i=1}^{f} d_i$, searching the two-tier structure once costs slightly more than $\frac{1}{16}$ of data pages in the worst case (i.e., traversing the whole two-tier structure). However, in a real situation, only a part of the structure is searched. Hence I/O_R and I/O_{BS} are expected to be less than $Page_R$ and $Page_{BS}$.

Recall that ASAP can adjust DIM' to further tighten the searching space by user-defined threshold on the number of candidates. A sequential I/O is usually a factor of 10 times faster then random access. Hence the number of candidates for random access may dominate the overall I/O cost. A large number of candidates are always undesirable. ASAP is iterative and parts of R-tree and bit sequence level are scanned in each iteration. In the worst case, the whole two-tier structure is scanned. To ensure ASAP is more efficient than sequential scan, the following relationship has to be true assuming that a sequential I/O is 10 times faster than random I/O:

$$\Phi * 10 + \alpha * (I/O_R + I/O_{BS}) \leq Page_{scan}$$

in the worst case, i.e., when the whole two-tier structures are searched, we have

$$\Phi \leq \frac{Page_{scan} - \alpha * Page_{scan}(\frac{3f}{\sum_{i=1}^{f} d_i} + \frac{1}{16})}{10}$$

where Φ and α represent the number of candidates and the number of iterations for a query to finish its KNN search.

Since usually $f \ll \sum_{i=1}^{f} d_i$, $\frac{3f}{\sum_{i=1}^{f} d_i}$ can be neglected. Consequently, we have

$$\Phi \leq \frac{(1 - \frac{\alpha}{16})}{10} * Page_{scan}$$

By enforcing the above inequality, I/O cost can be controlled so as to outperform sequential scan.

6.4 Performance Study

In this section, we report the results of an extensive performance study conducted to evaluate the proposed strategy on both real and synthetic data sets. We also compare against three scenarios: a single high-dimensional index structure that indexes on all dimensions of all features, a single index per feature, and sequential scan.

6.4.1 Experiment SetUp

All the experiments were performed on a Sun UtraSparc II 450Mhz (2 CPU), with 4G memory. We used a page size of 4K. All results reported are based on 50 queries for 10NN. The queries are data points obtained from the respective databases used. We adopted the L_2 norm for indexing and searching. All our experiments were conducted using the following data sets:

- WWW Image Features: We created a dataset of 73,715 WWW images extracted from over 40,000 web sites reached by the crawler in ICICLE system. This dataset consists of two feature sets. One is a 159-dimensional Color Histogram extracted from each image. The other is a 64-dimensional Daubechies' Wavelet [58] feature. Wavelet features describe an image's shape, texture and location information in a single representation. We truncated the wavelet representation to the 64 most dominant wavelet coefficients. The Color Histogram was normalized by d_{min}=0 and d_{max}=0.5, and the Wavelet feature was normalized by d_{min}=0 and d_{max}=8.

- Corel Image Features: This dataset contains image features extracted from a Corel image collection with 62,480 images [1]. It consists of four sets of features. The first is a 32-dimensional HSV Color Histogram. The second is

a 32-dimensional Color Histogram Layout. The third is a 16-dimensional Co-occurrence Texture feature. And the fourth is a 9-dimensional Color Moments feature. The first three sets were normalized by d_{min}=0 and d_{max}=0.5, and the last one was normalized by d_{min}=0 and d_{max}=3.

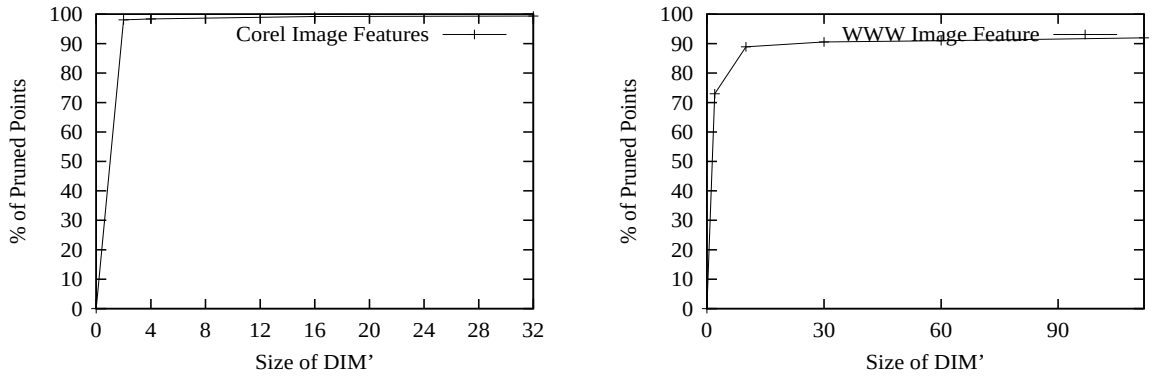
- Zipf Data:

  We created six synthetic data sets to study the effect of skew. Each set consists of 100,000 data points, and has 2 features of 64 dimensions. We denote them as Zipf(θ_1, θ_2), where θ_1 and θ_2 are the skew factor used in the Zipf distribution. This is done as follows. We first generate three 100,000 64-dimensional data points using the Zipf distribution with θ=1.0, 1.5 and 2.0 respectively. Next, we generate Zipf(θ_1, θ_2) by combining the two feature points generated by using skew factor of θ_1 and θ_2. In our experiments, we have used Zipf(1.0,1.0), Zipf(1.5,1.5), Zipf(2.0,2.0), Zipf(1.5,1.0), Zipf(1.5,1.5) and Zipf(1.5,2.0). All sets were normalized by d_{min}=0 and d_{max}=8.

6.4.2 Insight of DIM'

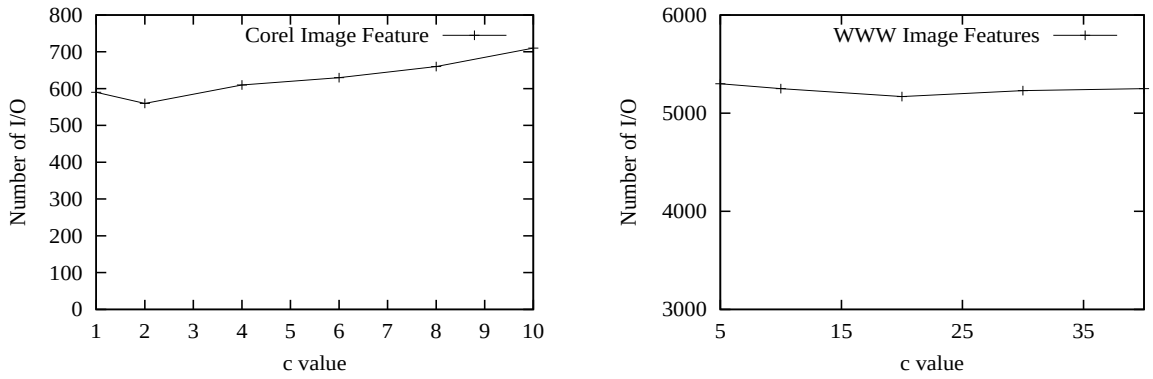
We first conduct experiments to have an insight on the effect of DIM'. We use the Corel dataset but restrict to just two features - HSV Color Histogram and Color Histogram Layout. Figure 6.6 depicts the effect of different size of DIM' where the x-axis represents the average number of dimensions for both features. Here, the two features are equally important. From Figure 6.6a, we see that with only two dimensions in DIM', ASAP can prune away more than 98% of the points. We repeated the experiment on WWW image dataset. The weights for Color Histogram and Wavelet features are set to 0.8 and 0.2 since Color Histogram has more dimensions. As shown in Figure 6.6b, with only ten dimensions in DIM', ASAP is able to prune away more

than 90% of the points. This experiment confirms that a small subset DIM' of the original dimensions will be sufficient for space pruning to avoid expensive full dimensional computation.



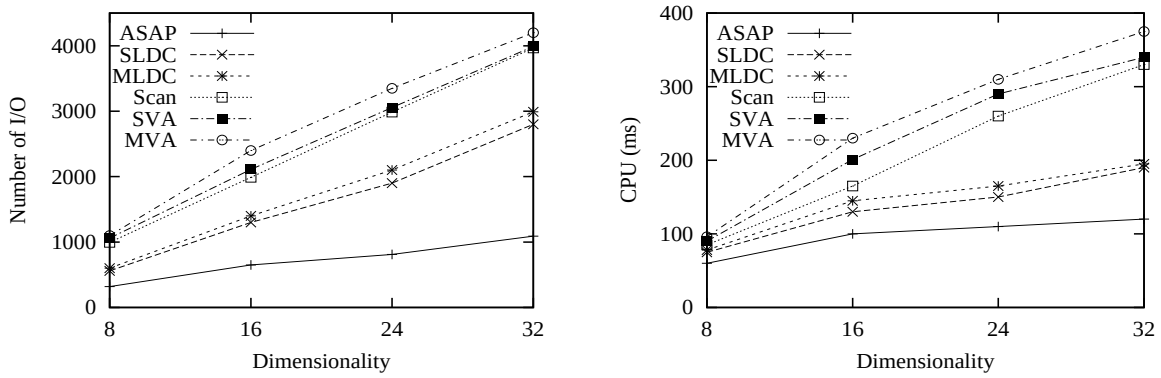
(a) On Corel Image Features (b) On WWW Image Features

Figure 6.6: Pruning Effect of DIM'.



(a) On Corel Image Features (b) On WWW Image Features

Figure 6.7: Effect of c.



(a) I/O Cost vs. Dimensionality

(b) CPU Cost vs. Dimensionality

Figure 6.8: Effect of Dimensionality on Corel Image Features.

6.4.3 Effect of c

Since the parameter c affects the number of intervals (which influences the construction of distance histogram), in this experiment, we study the effect of c. Figure 6.7 shows the effect of c for the above Corel and WWW image datasets' two-feature spaces. It is interesting to note that the performance of our scheme is not very sensitive to c. We believe this is mainly because different number of intervals for a feature vector will not greatly change the pattern of its distance histogram. Hence a c value can be easily selected. In the subsequent experiments, we set c to 2 and 20 for Corel and WWW image features respectively.

6.4.4 Effect of Dimensionality

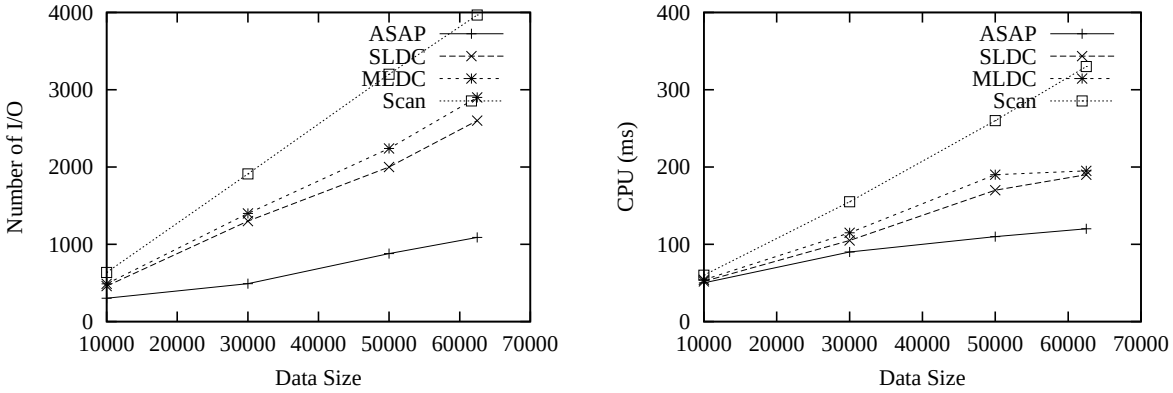
In this experiment, we study the effect of dimensionality. Here we tested the Corel images' two-features space used above, and both features have the same number of dimensions which we vary from 8 to 32. We shall use ASAP to denote the proposed KNN search scheme on the proposed two-tier indexing structure. We compare our method with two versions of LDC one single index on multiple features (called SLDC)

and one index per feature (MLDC), sequential scan and two versions of the VA-file, one single index on multiple features (called SVA) and one index per feature (MVA). The results, shown in Figure 6.8, clearly show that both the I/O and CPU cost for the six methods increase with the number of dimensions. From Figure 6.8a, we observe that ASAP performs the best, followed by SLDC, MLDC, sequential scan, SVA and MVA. It is evident that the VA-file methods examine most points for data level access, which causes the I/O cost to be even worse than sequential scan. Employing multiple indexes with the VA-file further deteriorates the performance by searching multiple indexes. The LDC methods outperforms sequential scan because it has ability to control the number of candidates as explained in previous chapter.

However, the proposed ASAP scheme outperforms SLDC and MLDC by a factor of 2-3 and outperforms sequential scan by a factor of 3-4. In fact, the gain of ASAP over the rest becomes larger as dimensionality goes up. The figure also shows that ASAP is robust to the size of the dimensions. Figure 6.8b depicts similar trend. In view of the poor performance of VA-files, in all subsequent experiments, we will compare our scheme with SLDC, MLDC and sequential scan only.

6.4.5 Effect of Data Size

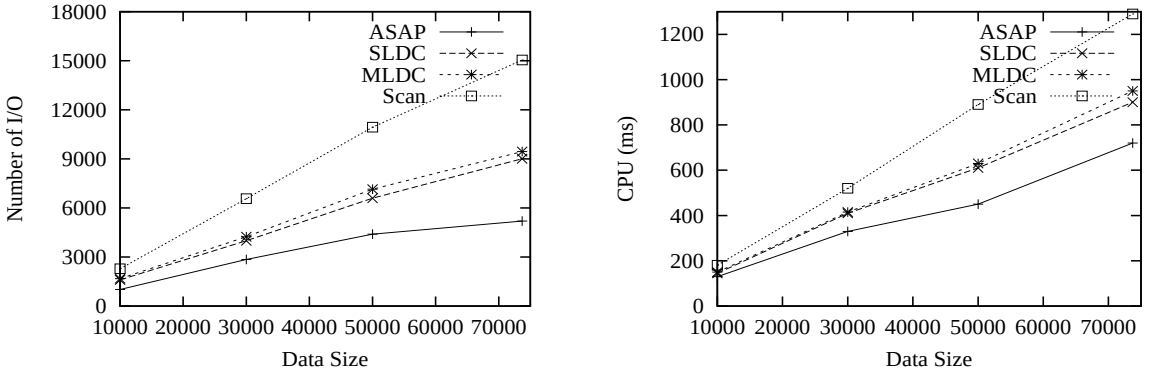
Next, we tested the effect of data size on our method using the Corel's and WWW's two-feature spaces mentioned above. For each dataset, we randomly picked subsets of different sizes (from 10000 onwards). Figures 6.9 and 6.10 show the results. As expected, the I/O and CPU costs increase as the data size increases. We also observe that both figures (on different datasets) have similar trends. While the costs for SLDC, MLDC and sequential scan increase linearly, the costs for ASAP increase in a much slower rate, i.e., the gain achieved by ASAP becomes more significant with a larger data size.



(a) I/O Cost vs. Data Size

(b) CPU Cost vs. Data Size

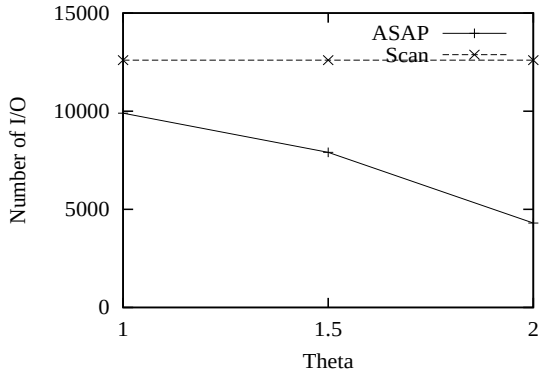
Figure 6.9: Effect of Data size on Corel Image Features.



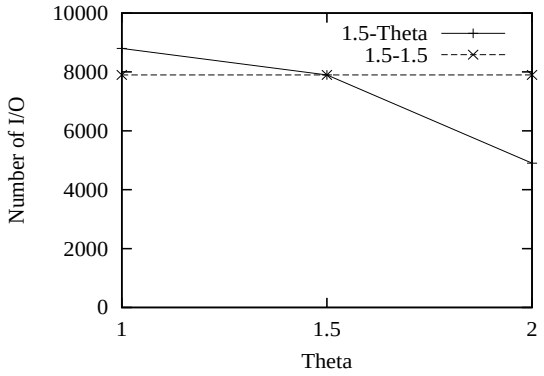
(a) I/O Cost vs. Data Size

(b) CPU Cost vs. Data Size

Figure 6.10: Effect of Data size on WWW Image Features.



(a) Two Concurrently varying *theta*



(b) One varying *theta* while the other unchanged

Figure 6.11: Effect of Skew.

6.4.6 Effect of Skew

Because of its adaptive threshold, we believe that one of the greatest strengths of ASAP is its ability to handle skew. This is so since data skew will keep the size of DIM' small and result in more informative bit encodings for each such dimension. We saw some evidence of this in the real data sets used in previous experiments. However, to study the effect of skew in isolation, we used the synthetic Zipf(θ_1, θ_2) data sets that we generated. Figure 6.11a shows the I/O cost for using the datasets where $\theta_1 = \theta_2$. As expected, data skew favors the performance of ASAP. As the data becomes more skewed, ASAP achieves better efficiency.

To see the skew effect more clearly, we repeated the experiments with data sets Zipf(θ_1, θ_2) where θ_1 is fixed at 1.5 while θ_2 is varied from 1.0 to 2.0. As shown in Figure 6.11b, ASAP achieves better performance for data that is more skewed. In real life skew datasets such as our Corel and WWW image features, we can expect good results. Now the reason for better performance on Corel features gained by ASAP is clear. Both Corel features are Color Histograms. However, one feature in

WWW image is texture which is much less skew than Color Histogram. Since ASAP performs better in a skew environment, it is not surprising that ASAP achieves greater improvement on Corel features.

6.4.7 Effect of Weighted Queries

In this experiment, we examine effect of weights that queries placed on different features. We used the Zipf(2.0,2.0) data set here. Figure 6.12 shows the corresponding I/O cost when the weight of one feature changes from 0.1 to 0.9 while the sum of the two weights is kept constant at 1.0. As we can see, ASAP achieves the best performance when the difference between two weights is the largest (comparing to cost when both features have same weights). Clearly, when one feature's weight is much larger than the other, it implicitly puts further skewness on the more important feature. Hence it is expected to lead to even better performance when the dominating feature is further skewed.

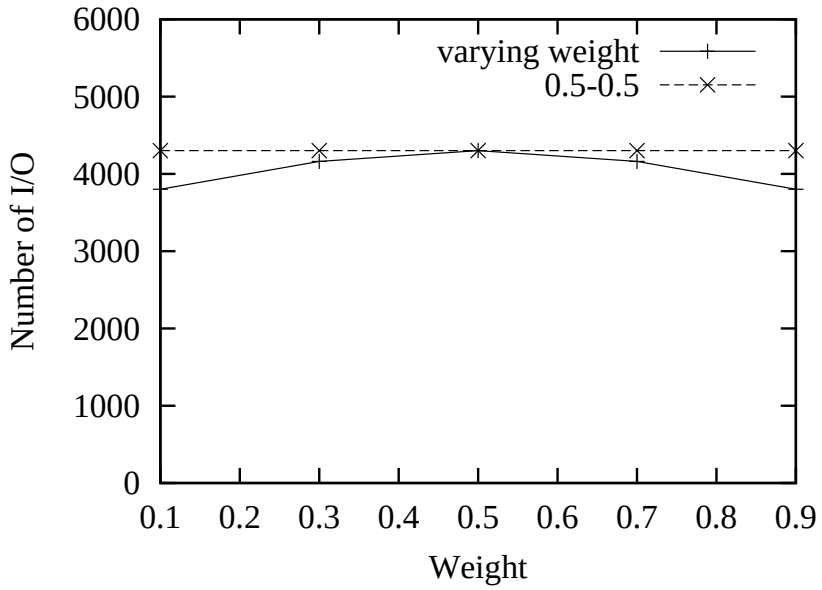


Figure 6.12: Effect of Weighted Queries.

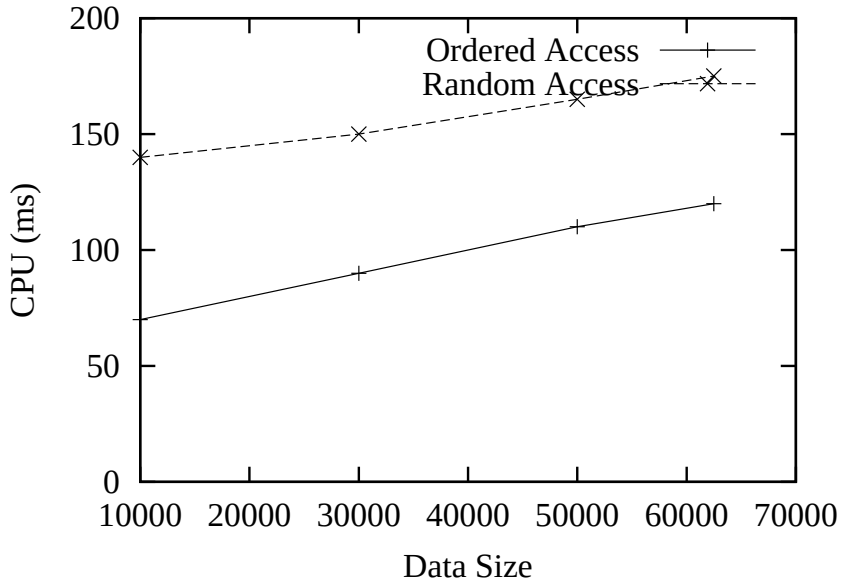
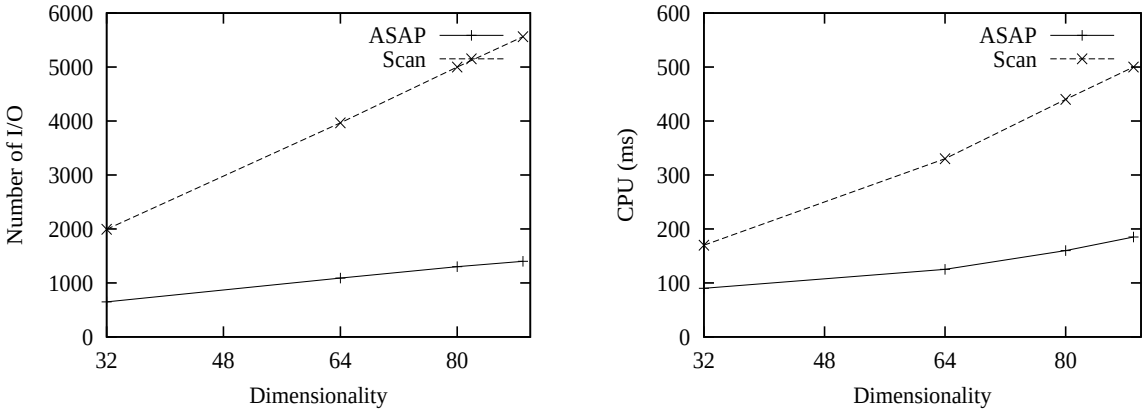


Figure 6.13: Effect of Access Order on Corel Feature



(a) I/O Cost

(b) CPU Cost

Figure 6.14: Effect of Number of Features.

6.4.8 Effect of Access Order

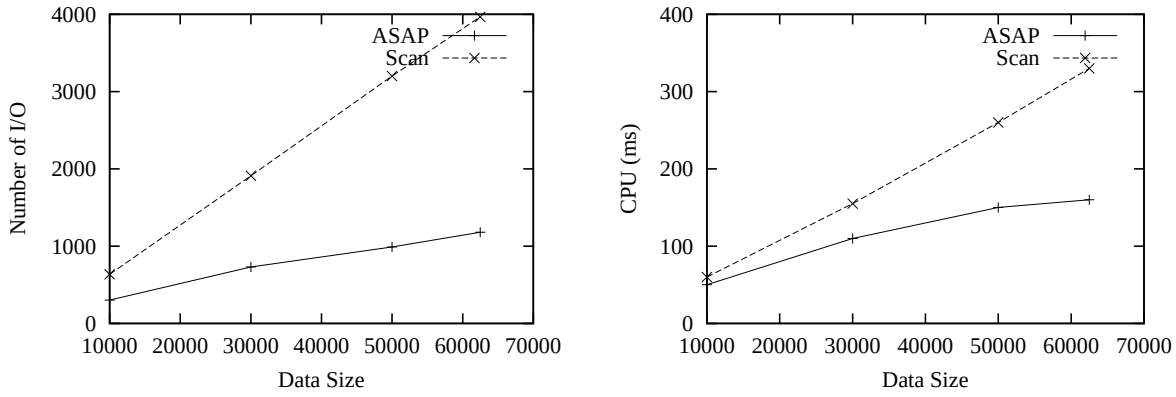
Recall that ASAP can optimize the feature access order by comparing the query features' RICs. It can avoid the extra computational cost by pruning the points in the earlier stage. Figure 6.13 compares effects of ordered access by RIC and random access on the CPU cost by using the Corel two-features space. We can see that ordered access in ASAP always outperforms random feature access greatly. This experiments shows the importance of selecting the right access order.

6.4.9 Effect of Number of Features

In this experiment, we test the effect of different number of features in our indexing structure. We use the Corel image's four features: Color Histogram, Color Histogram Layout, Co-occurrence Texture and Color Moments, and keep adding each feature one by one. Figure 6.14 shows the corresponding costs after each feature is added. We can see the the performance of ASAP is independent of the number of features. The cost increases almost linearly with the total dimensionality and has a smaller slope than sequential scan.

6.4.10 Effects of Dynamic Insertion

In this experiment, we test the effect of dynamic insertion on the efficiency of our single indexing structure. We used the Corel's two-feature dataset. We constructed the index using the first 10,000 images and then we inserted 20,000 images at a time. After each insertion, we performed KNN search and computed their average costs. Figure 6.15 shows the changing trends of I/O cost and CPU cost for ASAP and sequential scan as more images are inserted. It is clear that a larger number of images correspond to a larger index which naturally leads to larger I/O and CPU costs for



(a) I/O Cost

(b) CPU Cost

Figure 6.15: Effect of Dynamic Insertion on Corel Image Features.

KNN queries. Figure 6.15 depicts such trends. However, comparing with the linear increasing rate of sequential scan, the increasing rates of ASAP are much slower. And as more images are inserted, the increasing rate of ASAP becomes smaller and smaller.

Compared to the performance of ASAP with one-off construction given in Figure 6.9, we notice that the performance of ASAP slightly degrades by dynamic insertion. Recall that the two-component representation of an image is generated based on the center of data space. As more points arise, the center may move away from the original one. This scenario affects the effectiveness of our representation. Obviously, the larger distance the center moves away, the more negative effect on the performance of ASAP. To preserve the effectiveness of ASAP, one way is to compute the distance between the original center and the new center after insertion. Once the the center has shifted more than a distance allowed, the index need to be rebuilt.

6.5 Summary

In this chapter, we have reexamined the issue of efficient processing of multi-feature queries. We have devised a novel representation that compactly captures a f-feature point into a two vector component: an f-dimensional vector and a bit sequence. This representation leads to a two-level index structure where the first tier indexes the first component using a standard multi-dimensional index structure such as an R-tree, and the second level is a compact list of bit sequences accessible from the leaf node entries of the first level. We have also proposed an efficient algorithm for processing k-nearest neighbor queries. Our extensive experiments on both real life and synthetic datasets show that the our proposal offers significant advantages over existing methods.

Chapter 7

Conclusions

7.1 Contributions

In this thesis, we address the problem of efficient database support for effective WWW image retrieval. We proposed ICICLE, an effective semantic-based WWW image retrieval system. ICICLE is further extended to include multiple visual features. To achieve efficient database support for the extended ICICLE, we proposed three novel indexing techniques: Multi-level Mahalanobis-based Dimension Reduction (MMDR), Local Digital Coding (LDC), and the two-tier indexing structure. The MMDR was designed for high-dimensional feature with local correlations among dimensions. The LDC was designed for hyper-dimensional feature to break the 'dimensionality curse' by scaling the dimensionality to be thousands. And the two-tier structure was designed for indexing databases with multiple features. These techniques have been shown to be superior than existing indexing methods. As a result, by employing them, the extended ICICLE system achieves more efficient database support in query answering.

We are pleased to note that part of this research has been applied in a commercial image database management system (http://www.geofoto.com). And the research has resulted in a number of technical papers in image retrieval area [48, 49, 50] and

database area [30, 31, 33, 40].

7.1.1 Semantic-based Image Retrieval

To solve the issues involved in an effective and efficient WWW image search engine,we first introduced a new model to represent the content of images embedded in WWW pages. The proposed Weight ChainNet model combines different types of lexical chains obtained from the surrounding text of an image. Our experimental study showed that the approach can be used as an effective means to represent image semantics. We also proposed two novel feedback mechanisms. In particular, the semantic integration and differentiation method returned more accurate results than semantic accumulation with higher recall.

Moreover, we have also presented a new incremental clustering algorithm ICC for the increasingly growing large database collection of WWW images described by text information. Our experiments showed that ICC can produce quality clusters, and can adapt the cluster size and cluster number dynamically. Without looking at the actual data points, ICC can identify the sub-clusters within a large cluster by checking the cluster representative's property. It can also handle temporary noise well by using a special 'noise box' from which new clusters can be generated. To improve efficiency, ICC employs a summarization step called Vertical and Pyramidal Summarization Tree. VP-ST starts from all the data points in the cluster, and finally converges all summarized points into single representative of the cluster. Our experiments indicated that this vertical and pyramidal technique provides quality representative, especially compared with random sampling. To further speed up searching right cluster, the clusters' structure is in Hierarchical ChainNet Summarization tree. A prototype system, called ICICLE, that employs the proposed models have been deployed in our

VIPER project (http://sloth.comp.nus.edu.sg/m̃mir/). To integrate with Content-Based Image Retrieval (CBIR), ICICLE was also extended to consider low-level visual features.

7.1.2 High-dimensional Indexing

To support efficient retrieval for single image feature space, we have presented an effective and fast dimensionality reduction algorithm – Multi-level Mahalanobis-based Dimensionality Reduction (MMDR), which is able to reduce the number of dimensions while keeping the precision high, and able to effectively handle large datasets and dynamic insertions. We used a single structure to index the data points in different reduced subspaces. We conducted extensive experimental studies using both real and synthetic datasets to compare the algorithm with existing approaches. The results show that the proposed technique, as a whole, is very effective and efficient in supporting KNN search in very high-dimensional space. Furthermore, it is scalable for very large databases and able to hand dynamic insertions adaptively.

7.1.3 Hyper-dimensional Indexing

To support a feature space in hundreds or more dimensionality, we introduced a very effective data organization and representation methodology called Local Digital Code (LDC) suitable for hyper-dimensional data. Such representation encompasses the application of partial distance and accommodates a novel KNN search algorithm - SPA.

SPA uses the minimal partial distance computed from any m dimensions among n most informative dimensions between the query and static reference points (cluster centers), as the partial distance. Such partial distance computation avoids accessing

data points so that the overall computational costs are minimized. SPA is capable of pruning points in the data space rapidly, without computing distances among them, employing DCs and simple bitwise operations. Moreover, SPA can minimize the candidate point set that requires retrieval and further processing, by employing the results of our analytical methodology. Our extensive performance study on hyper-dimensional data demonstrated that SPA outperforms known methods significantly.

7.1.4 Multiple Feature Indexing

To fully support efficient image retrieval in multi-feature space, we have reexamined the issue of efficient processing of multi-feature queries. We have devised a novel representation that compactly captures a f-feature point into a two vector component: an f-dimensional vector and a bit sequence. This representation leads to a two-level index structure where the first tier indexes the first component using a standard multi-dimensional index structure such as an R-tree, and the second level is a compact list of bit sequences accessible from the leaf node entries of the first level. We have also proposed an efficient algorithm called ASAP for processing KNN mutli-feature queries. Our extensive experiments on both real life and synthetic datasets show that the proposed index structure offers significant performance advantages over existing methods.

7.2 Future Work

As the system contains issues in both image retrieval area and database area, our work can be extended in several ways. We plan to extend ICICLE in the following ways. First, since we are mainly concerned with the object and event, it may be helpful to guess the lexical chain meaning by applying AI techniques and extend

HTML documents to XML documents. We are currently looking into some of these techniques. Second, the proposed approach is essentially an Information Retrieval (text-based) approach. We plan to integrate with effective content-based retrieval methods that capture the visual content of the images, especially the shape. Third, WWW images could be updated frequently. In particular, some images may be removed. We are planning to look at some more operation like delete. Fourth, we plan to integrate our approach with access methods to further speed up the retrieval process. Finally, we might consider the integrated adaptive double clustering on both text representation and visual feature representations, with relevance feedback's semantics considered.

For the indexing techniques, they can be extended to many other new emerging applications. Notice that some emerging research areas, such as Bioinformatics, are shaping the current research and deal with data with thousands of dimensionality or much more. How to perform fast sequence matching and searching has become a new challenging. As a future work, methodologies in the spirit of LDC can be possibly extended for biology information retrieval to support fast access. Due to LDC's small size and fast comparison, it can also be further designed for summarization and indexing of data streams. As for the proposed two-tier structure, extended research is undergoing to solve the well-known problem called 'preference query'.

Bibliography

[1] *http://kdd.ics.uci.edu/databases/corelfeatures.*

[2] C.C. Aggarwal, A. Hinneburg, and D. A. Keim, *On the surprising behavior of distance metrics in high dimensional spaces*, ICDT, 2001, pp. 420–434.

[3] C.C. Aggarwal, J.L. Wolf, P.S. Yu, C. Procopiuc, and J. S. Park, *Fast algorithms for projected clustering*, SIGMOD, 1999, pp. 61–72.

[4] M. Ankerst, M.M. Breunig, H.-P. Kriegel, and J. Sander, *OPTICS: Ordering points to identify the clustering structure*, SIGMOD, 1999, pp. 49–60.

[5] Y.A. Aslandogan and C.T. Yu, *Evaluating strategies and systems for content based indexing of person images on the web*, ACM Multimedia, 2000, pp. 313–321.

[6] S. Berchtold, C. Böhm, H. V. Jagadish, H.-P. Kriegel, and J. Sander, *Independent quantization: An index compression technique for high-dimensional data spaces*, ICDE, 2000, pp. 577–588.

[7] S. Berchtold, C. Böhm, and H.-P. Kriegel., *The pyramid-technique: Towards breaking the curse of dimensionality*, SIGMOD, 1998, pp. 142–153.

[8] S. Berchtold and D. A. Keim, *Indexing high-dimensional spaces: Database support for next decade's applications*, ACM Computing Surveys **33(3)** (2001), 322–373.

[9] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, *When is nearest neighbors meaningful?*, ICDT, 1999, pp. 217–235.

[10] A. Borodin, R. Ostrofsky, and Y. Rabani, *Subquadratic Algorithms for Approximate Clustering in High Dimensional Spaces*, ACM STOC, 1999.

[11] G.H. Cha and C. W. Chung, *An indexing and retrieval mechanism for complex similarity queries in image databases*, Journal of Visual Communication and Image Representation **10(3)** (1999), 268–290.

[12] G.H. Cha and C.W. Chung, *An indexing and retrieval mechanism for complex similarity queries in image databases*, Journal of Visual Communication and Image Representation **10(3)** (1999), 268–290.

[13] K. Chakrabarti and S. Mehrotra, *Local dimensionality reduction: A new approach to indexing high dimensional spaces*, VLDB, 2000, pp. 89–100.

[14] K. Chakrabarti and S.Mehrotra, *The hybrid tree: An index structure for high dimensional feature spaces*, ICDE, 1999, pp. 322–331.

[15] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, *Incremental clustering and dynamic information retrieval*, ACM Sympos. on Theory of Computing, 1997, pp. 626–634.

[16] T.S. Chua, W.C. Low, and C.X. Chu, *Relevance feedback techniques for color-based image retrieval*, Multimedia Modelling, 1998, pp. 24–31.

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

[18] Essam A. El-Kwae and Mansur R. Kabuka, *Efficient content-based indexing of large image databases*, ACM Transactions on Information Systems **18(2)** (2000), 171–210.

[19] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu, *Incremental clustering for mining in a data warehousing environment*, VLDB, 1998, pp. 323–333.

[20] R. Fagin, A. Lotem, and M. Naor, *Optimal aggregation algorithms for middleware*, PODS, 2001, pp. 102–113.

[21] C. Frankel, M. J. Swain, and V. Athitsos, *Webseer: An image search engine for the world-wide web*, U. of Chicago Tech. rep. TR 96-14, 1996.

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

[23] A. Gionis, P. Indyk, and R. Motwani, *Similarity search in high dimensions via hashing*, VLDB, 1999, pp. 518–529.

[24] S. Guha, R. Rastogi, and K. Shim, *CURE: An Efficient Clustering Algorithm for Large Databases*, SIGMOD, 1998, pp. 73–84.

[25] U. Guntzer, W-T. Balke, and W. Kiessling, *Optimizing multi-feature queries for image databases*, VLDB, 2000, pp. 261–281.

[26] V. Harmandas, M. Sanderson, and M.D. Dunlop, *Image retrieval by hypertext links*, ACM SIGIR, 1997, pp. 296–303.

[27] M.A. Hearst and J.O. Pedersen, *Reexamining the cluster hypothesis: Scatter/gather on retrieval results*, ACM SIGIR, 1996, pp. 76–84.

[28] A. Hinneburg and D. A. Keim, *An optimal grid-Clustering: Towards breaking the curse of diminsionality in high dimensional clustering*, VLDB, 1999, pp. 506–517.

[29] A. Hinneburg and D.A. Keim, *An efficient approach to clustering in large multimedia databases with noise*, SIGKDD, 1998.

[30] H.V. Jagadish, B.C. Ooi, H.T. Shen, and K.L. Tan, *Towards Efficient Multi-Feature Query Processing*, 2003,Submitted for Publication.

[31] H. Jin, B.C. Ooi, H.T. Shen, C. Yu, and A. Zhou, *An Adaptive and Efficient Dimensionality Reduction Algorithm for High-Dimensional Indexing*, ICDE, 2003, pp. 87–98.

[32] I.T. Jolliffe, *Principal component analysis*, Springer-Verlag, 1986.

[33] N. Koudas, B.C. Ooi, H.T. Shen, and A. Tung, *LDC: Enabling Search By Partial Distance In A Hyper-Dimensional Space*, ICDE, 2004.

[34] J.H. Lee, D.H. Kim, and C.W. Chung, *Multi-dimensional selectivity estimation using compressed histogram information*, SIGMOD, 1999, pp. 205–214.

[35] G.A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K.J. Miller, *Introduction to wordnet: An on-line lexical database*, International Journal of Lexicography (special issue) **3(4)** (1990), 235–312.

[36] J. Morris and G. Hirst, *Lexical cohesion computed by thesaural relation and an indicator of the structure of text*, Computational Linguistics **17(1)** (1991), 22–48.

[37] S. Mukherjea, K. Hirata, and Y. Hara, *Amore: A world wide web image retrieval engine*, World Wide Web **2(3)** (1999), 115–132.

[38] Anne H.H. Ngu, Q.Z. Sheng, D.Q. Huynh, and R. Lei, *Combining multi-visual features for efficient indexing in a large image database*, VLDB Journal **9(4)** (2001), 279–293.

[39] W. Niblack, R. Barber, and W. Equitz, *The qibc project: querying images by content using color, texture, and shape*, Technical report, IBM RJ 9203(81511), 1993.

[40] B.C. Ooi, H.T. Shen, and C. Xia, *Towards Efficient Multi-feature Based WWW Image Retrieval*, 4th Pacific-Rim Conference On Multimedia, 2003 (Invited Paper).

[41] B.C. Ooi, K.L. Tan, T.S. Chua, and W. Hsu, *Fast image retrieval using color-spatial information*, VLDB Journal **7(2)** (1998), 115–128.

[42] B.C. Ooi, K.L. Tan, C. Yu, and S. Bressan, *Indexing the edges: A simple and yet efficient approach to high-dimensional indexing*, PODS, 2000, pp. 166–174.

[43] C.R. Palmer and C. Faloutsos, *Density biased sampling: an improved method for data mining and clustering*, SIGMOD, 2000, pp. 82–92.

[44] G. Pass, R. Zabin, and J. Miller, *Computing images using color coherence vectors*, ACM Multimedia, 1996, pp. 65–73.

[45] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima, *The A-tree: An index structure for high-dimensional spaces using relative approximation*, VLDB, 2000, pp. 516–526.

[46] S. Sclaro, L. Taycher, and M.L. Cascia, *Imagerover: A content-based image browser for the world wide web*, IEEE Workshop on Content-Based Access of Image and Video Libraries, 19967.

[47] G. Sheikholeslami, S. Chattajee, and A. Zhang, *WaveCluster: a multi-resolution clustering approach for Very Large Spatial Databases*, VLDB, 1998.

[48] H.T. Shen, *Finding similar images quickly using object shapes*, ACM CIKM, 2001, pp. 498–506.

[49] H.T. Shen, B.C. Ooi, and K. L. Tan, *Finding semantically related images in WWW*, ACM Multimedia, 2000, pp. 491–493.

[50] H.T. Shen, B.C. Ooi, and K.L. Tan, *Giving meanings to WWW images*, ACM Multimedia, 2000, pp. 39–48.

[51] A. F. Smeaton and I. Quigley, *Experiments on using semantic distances between words in image caption retrieval*, ACM SIGIR, 1996, pp. 174–180.

[52] J.R. Smith and S. F. Chang, *Visually searching the web for content*, IEEE Transaction on Multimedia **4(3)** (1997), 12–20.

[53] J.R. Smith and S.F. Chang, *Image indexing and retrieval based on color histograms*, ACM Multimedia, 1996, pp. 87–98.

[54] R.K. Srihari., *Automatic indexing and content-based retrieval of captioned images*, IEEE Computer **28(9)** (1995), 49–56.

[55] K.K. Sung and T. Poggio, *Example-based learning for view-based human face detection*, PAMI **20(1)** (1998), 39–51.

[56] J.S. Vitter and M. Wang, *Approximate computation of multidimensional aggregates of sparse data using wavelets*, SIGMOD, 1999, pp. 193–204.

[57] E.M. Voorhees, *Query expansion using lexical-semantic relations*, ACM SIGIR, 1994.

[58] J. Z. Wang, G. Wiederhold, O. Firschein, and S. X. Wei, *Content-based image indexing and searching using daubechies wavelets*, Int'l Journal of Digital Libraries **1(4)** (1998), 311–328.

[59] R. Weber, H. Schek, and S. Blott, *A quantitative analysis and performance study for similarity search methods in high dimensional spaces*, VLDB, 1998, pp. 194–205.

[60] S. K.M. Wong, W. Ziarko, V. V. Raghavan, and P. C.N. Wong, *On modeling of information retrieval concepts in vector spaces*, TODS, 1987.

[61] C. Yu, *High-dimensional indexing*, Ph.D. thesis, Department of Computer Science, National University of Singapore, 2001.

[62] C. Yu, B.C. Ooi, K.L. Tan, and H. V. Jagadish, *Indexing the distance: An efficient method to KNN processing*, VLDB, 2001, pp. 166–174.

[63] T. Zhang, R. Ramakrishnan, and M. Livny, *BIRCH: An Efficient Data Clustering Method for Very Large Databases*, SIGMOD, 1996, pp. 103–114.

[64] C. Zheng, W. Liu, C. Hu, M. Li, and H. Zhang, *ifind: A web image search engine*, ACM SIGIR (demo), 2001, p. 450.

[65] C. Zheng, W. Liu, F. Zhang, M. Li, and H. Zhang, *Web mining for web image retrieval*, Journal of the American Society for Information Science and Technology **52(10)** (2001).