EFFICIENT INDEX STRUCTURES FOR VIDEO DATABASES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ESRA AÇAR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

JANUARY 2008

Approval of the thesis:

EFFICIENT INDEX STRUCTURES FOR VIDEO DATABASES

submitted by **ESRA AÇAR** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Volkan Atalay _____
Head of Department, **Computer Engineering**

Prof. Dr. Adnan Yazıcı _____
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. İsmail Hakkı Toroslu _____
Computer Engineering Dept., METU

Prof. Dr. Adnan Yazıcı _____
Computer Engineering Dept., METU

Assoc. Prof. Dr. Ahmet Coşar _____
Computer Engineering Dept., METU

Asst. Prof. Dr. Murat Koyuncu _____
Computer Engineering Dept., Atılım University

M.S. Serdar Arslan _____
Computer Center, METU

Date: 29.01.2008

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Esra AÇAR

Signature :

ABSTRACT

EFFICIENT INDEX STRUCTURES FOR VIDEO DATABASES

Açar, Esra
M.S., Department of Computer Engineering
Supervisor: Prof. Dr. Adnan YAZICI

January 2008, 130 pages

Content-based retrieval of multimedia data has been still an active research area. The efficient retrieval of video data is proven a difficult task for content-based video retrieval systems. In this thesis study, a Content-Based Video Retrieval (CBVR) system that adapts two different index structures, namely Slim-Tree and BitMatrix, for efficiently retrieving videos based on low-level features such as color, texture, shape and motion is presented. The system represents low-level features of video data with MPEG-7 Descriptors extracted from video shots by using MPEG-7 reference software and stored in a native XML database. The low-level descriptors used in the study are Color Layout (CL), Dominant Color (DC), Edge Histogram (EH), Region Shape (RS) and Motion Activity (MA). Ordered Weighted Averaging (OWA) operator in Slim-Tree and BitMatrix aggregates these features to find final similarity between any two objects. The system supports three different types of queries: exact match queries, k-NN queries and range queries. The experiments included in this study are in terms of index construction, index update, query response time and retrieval efficiency using ANMRR performance metric and precision/recall scores. The experimental results show that using BitMatrix along with Ordered Weighted Averaging method is superior in content-based video retrieval systems.

Keywords: Content-Based Video Retrieval, MPEG-7, Color Layout, Dominant Color, Edge Histogram, Region Shape, Motion Activity, Slim-Tree, BitMatrix, Ordered Weighted Averaging, XML Database

ÖZ

VİDEO VERİTABANLARI İÇİN ETKİLİ İNDEKS YAPILARI

Açar, Esra
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Adnan YAZICI

Ocak 2008, 130 sayfa

İçerik-tabanlı multimedya erişimi halen aktif bir araştırma alanıdır. İçerik-tabanlı video erişim sistemleri için video verisine hızlı ve verimli erişmenin zor bir işlem olduğu kanıtlanmıştır. Bu tez çalışmasında, renk, doku, şekil ve hareket gibi alt-düzey özellikleri temel alarak videolara etkili erişmek için Slim-Tree ve BitMatrix olarak adlandırılan iki farklı indeks yapısını adapte eden bir İçerik-Tabanlı Video Erişim (ITVE) sistemi sunulmaktadır. Sistem video verisine ait alt-düzey özelliklerini, MPEG-7 referans yazılımı kullanılarak video parçalarından elde edilen ve XML bir veritabanında saklanan MPEG-7 tanımlayıcıları ile belirtmektedir. Çalışmada kullanılan alt-düzey tanımlayıcılar Renk Planı, Baskın Renk, Kenar Dağılımı, Alan Şekli ve Hareket Aktivitesi'dir. Bu tanımlayıcılar Sıralı Ağırlıklı Ortalama operatörü kullanılarak Slim-Tree ve BitMatrix yapılarında herhangi iki nesne arasındaki benzerlik değerini bulmak için birleştirilir. ITVE sistemi tam eşleme, k-NN ve alan sorguları olmak üzere üç farklı tip sorgulamayı destekler. Bu çalışmada yer alan testler, indeks oluşturulması, indeks güncelleme, sorgu yanıt süresi ve ANMRR performans metrik değeri ve doğruluk/geriçağrım değerleri kullanılarak erişim etkinliğinin belirlenmesini kapsamaktadır. Yapılan testlerin sonuçları BitMatrix yapısının Sıralı Ağırlıklı Ortalama metodu ile beraber kullanılması durumunun içerik-tabanlı video erişim sistemlerinde en iyi sonucu vermekte olduğunu göstermektedir.

Anahtar Kelimeler: İçerik-Tabanlı Video Erişimi, MPEG-7, Renk Planı, Baskın Renk, Kenar Dağılımı, Alan Şekli, Hareket Aktivitesi, Slim-Tree, BitMatrix, Sıralı Ağırlıklı Ortalama, XML Veritabanı

To My Family…

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my thesis supervisor Prof. Dr. Adnan Yazıcı for his valuable guidance, motivation and support throughout this thesis study.

My special thanks are to Serdar Arslan for his guidance, criticism and insight throughout the research.

Finally, I want to send all my love to my family, Abdullah, Ceyda, Alpertan, Nurşen, Ural, Yalın and Şeyda Açar and wish to thank them for their support throughout all these years.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ANMRR	:	Average Normalized Modified Retrieval Rank
ARFF	:	Attribute-Relation File Format
AV	:	Audio-Visual
AVR	:	Average Rank
CBR	:	Content-Based Retrieval
CBVR	:	Content-Based Video Retrieval
CL	:	Color Layout
ct	:	Cardinality Threshold
DB	:	Database
DC	:	Dominant Color
DDL	:	Description Definition Language
Ds	:	Descriptors
DSs	:	Description Schemes
EH	:	Edge Histogram
EQW	:	Equal Weighted
et	:	Expansion Threshold
HTD	:	Homogenous Texture Descriptor
k-NN	:	k Nearest Neighbor
MA	:	Motion Activity
MBR	:	Minimum Bounding Rectangle
MPEG	:	Moving Pictures Experts Group
MRR	:	Modified Retrieval Rank
NMRR	:	Normalized Modified Retrieval Rank
OWA	:	Ordered Weighted Averaging
QBE	:	Query-By-Example
QBIC	:	Query By Image Content
QBS	:	Query-By-Subject
RS	:	Region Shape
STRG	:	Spatio-Temporal Region Graph
TB	:	Texture Browsing

VS	:	Video Shot
XM	:	Experimentation Model
XML	:	Extensible Markup Language
XXL	:	Extensible and Flexible Library

CHAPTER 1

INTRODUCTION

Multimedia data are of high importance in many areas and the growth for multimedia brings the need for more effective methods for content-based retrieval. The video is different from other multimedia data. Because video is:

- *Continuous*: consists of sequential frames where each frame can be thought as an image,
- *Time dependant*: it has a temporal aspect, and
- *Heterogeneous*: the richest type of media (image, audio and movement combined)

The content of video can be seen from three different perspectives [14]:

- *Textual content*: Data that is not directly concerned with the content of video data but in some way related to it such as date, movie title, director etc. [14].
- *Semantic content*: The idea or knowledge that convey to human (usually ambiguous, subjective, and context-dependant) when video is perceived by human.
- *Audiovisual content*: Low-level features of video such as color, texture, shape, motion and audio.

Due to the complexity inherent to video data, content-based video retrieval requires multidisciplinary research effort in areas such as image and video processing, machine learning, data mining, computer vision, visual data modeling, human visual perception, information retrieval and multimedia database systems.

Effective content-based indexing and retrieval of videos is proven be a difficult task in content-based video retrieval systems, because of the complexity of video data. Hence, our motivation in this thesis study is to construct a content-based video retrieval system that

utilizes Slim-Tree and adapts BitMatrix index structures along with OWA operator for effective content-based video retrieval.

Three main issues discussed in content-based video retrieval systems are [45]: (1) how to efficiently parse a sequential video stream into meaningful smaller video shots, (2) how to compute the similarity between any two video shots more accurately, and (3) how to index and retrieve video shots more effectively.

(1) Indexing low-level features of for example a one-hour length video does not make sense, since the queries are usually made for more specific events that occur in video such as video parts that anchorman/anchorwoman appears or that show crowded places etc. Thus, in content-based video retrieval systems, sequential video streams are broken into manageable units, namely video shots, whose content is similar as a whole.

(2) Content-based video retrieval systems use distance functions to compute the similarity between any two video shots. Hence, the distance functions used in the system are important elements of the system for effective indexing and retrieval.

(3) Due to the complexity inherent to video data, when the number of dimensions of feature vectors and/or the number of videos to be indexed increase, sequential scan of the video database becomes inefficient in terms of query response times. Therefore, efficient index structures that are scalable to large video databases and support indexing of multi-dimensional feature vectors of video are needed.

In the literature, many techniques and systems have been proposed in the context of multi-dimensional indexing. The common goal of these multi-dimensional indexing methods is to divide the search space in a set of ranges and prune some of the ranges at search time [1]. Metric-space methods such as studies introduced in [13] and [3] index the relative distances between the multi-dimensional feature vectors of objects, where vector-space methods such as studies introduced in [23], [24], [25] and [26] directly index the multi-dimensional feature vectors. Studies introduced in [19] and [22] propose mapping the multi-dimensional feature vectors into one-dimensional space and then use one of the efficient methods for one-dimensional indexing. Finally, studies introduced in [1], [20] and [21] state that sequential scan is inevitable and propose using an approximation file for feature vectors, pruning the approximations during queries based on their

minimum/maximum distance to the query object and computing the exact distances between the query object and the remaining objects in the database.

Some of the proposed index structures such as M-Tree [13] and BitMatrix [1][2] are used to index image data. The index structures used for efficient image indexing may be candidates for video indexing. However, video data is more complex than image data; in addition to still image features such as color, texture and shape, video has also motion features. Thus, the time-dependant nature and the motion properties of the video data are of considerable importance in choosing adequate video indexing techniques. A multi-dimensional index structure that is adequate for video shall be capable of expressing the motion features of video effectively. Studies introduced in [12] and [41] show that combining multiple features in content-based multimedia retrieval improves the query performance in terms of retrieval efficiency. Hence, the index structure used for video indexing shall also be able to combine the visual features of video data such as color, texture, shape and motion to improve the quality of query results.

There are two main approaches in content-based video retrieval [14]:

- *Low-level Systems* that retrieves video data based on low-level features of video such as color, texture, shape and motion. Studies introduced in [41] and [42] are examples of this type of CBVR systems. In low-level systems, the content of low-level descriptors that defines the low-level features of video such as color, texture, shape and motion are the key factors for the retrieval efficiency of the system. For example, a human percepts the objects firstly according to their shapes and if the shape descriptor used in the system can define the shape of objects similar to the perception of human, this increases the retrieval efficiency of the system. At this point, a common format that provides fast and effective retrieval helps to represent the low-level features of multimedia data. As a consequence, MPEG-7 [5], formally known as Multimedia Content Description Interface, is introduced as an ISO/IEC standard by MPEG (Moving Picture Experts Groups) for representing the audio-visual content by using standard descriptors. MPEG-7 provides visual descriptors for color, texture, shape and motion features of video that allow efficient content-based retrieval of video data.

- *High-level Systems* that intent to manage video data as it is perceived by human. In general, this type of systems defines a video data model that models the video

objects and the relations between them along the time axis and provides appropriate query languages to query video data. Studies introduced in [14] and [15] are examples of this type of CBVR systems. Low-level features of video data can be extracted automatically. However, automatic object recognition, tracking and classification are difficult problems. Hence, in high-level systems, the problem is the mapping between the low-level features of video data and semantic concepts appear in video. A simple way of this mapping is to annotate video manually. However, since this solution needs human-interaction with the system, it is inefficient for large video databases. Another way of this mapping may be using a knowledge-based system. Thus, by using a knowledge database for content-based video retrieval, the system can map high-level video query conditions such as keywords on particular subject(s) to the low-level features of video such as color, texture, shape and motion.

In this thesis study, we present a content-based retrieval system that utilizes Slim-Tree and adapts BitMatrix along with OWA operator for effective content-based indexing and retrieval of videos. The main contributions of this thesis study and the focused parts of a content-based retrieval system are as follows:

1- *Video Shot Detection and Keyframe Extraction* – First issue to focus is the segmentation of sequential video streams into video shots and the extraction of keyframes for the video shots. In this study, video shot detection and keyframe extraction are done by using IBM VideoAnnEx Annotation Tool [32]. These video shots and keyframes are stored in the file system for further processing.

2- *Feature Extraction and Storage* – Second issue to focus is the extraction process of low-level features of video data. In order to describe the low-level features, we use MPEG-7 visual descriptors. To describe the content of video data, the MPEG-7 visual descriptors used in this study are Color Layout (CL), Dominant Color (DC), Edge Histogram (EH), Region Shape (RS) and Motion Activity (MA). To extract the color, texture and shape low-level features of a video shot, keyframes of the video shots are used, where for the extraction of motion low-level features of video shots; video shots are used as a whole. These visual descriptors are extracted in XML format by using MPEG-7 eXperimentation Model (XM) [27] and stored in a

native XML database system called Oracle Berkeley DB [28] for further processing and retrieval.

3- *Video Database Indexing* – As mentioned before, sequential scan of the video database becomes inefficient when the number of database objects and/or the dimensions of the feature vectors increase. Thus, we need index structures for effective retrieval in terms of time and efficiency. In this study, two index structures, namely Slim-Tree [3] and BitMatrix [1][2] are presented in the system to improve the query performance in terms of time and efficiency. Similarity measurement for both structures is carried out by using a metric distance function called Euclidean Distance.

   In this study, BitMatrix is used for content-based video indexing for the first time. BitMatrix clusters feature vectors to generate feature vector approximations that are used for pruning during queries based on their minimum/maximum distance to the query object. This pruning strategy based on clustering results increases the retrieval performance of the system in terms of both time and retrieval efficiency. In addition to BitMatrix adaptation, Slim-Tree that has better performance than M-Tree according to performance tests in [3] is also utilized in the system by using an API, namely XXL API [29].

4- *Using Ordered Weighted Averaging (OWA) for Feature Aggregation* – In general, content-based retrieval systems combine features of video data by using fixed weights for each future. Thus, features of all videos in the database are associated with fixed weights during distance computation. However, when comparing two videos one feature may be more distinctive than other features; therefore that feature must be associated with higher weight. Thus, we adapt OWA [35] operator to the system to associate variable weights with the low-level features of video data (CL, DC, EH, RS and MA) when combining distance values of each feature into one final distance between any two objects.

5- *Query Module* – In content-based video retrieval, the characteristics of video you are looking for cannot be precisely defined, unlike in traditional SQL queries. Thus, content-based video retrieval system shall support similarity queries on videos. In this thesis study, Query-By-Example (QBE) paradigm is used for similarity queries on video data; the system searches low-level visual features stored in XML database to retrieve video shots that are similar to the given query video shot.

BitMatrix and Slim-Tree are used in the query module of the system to support efficient exact match, k-NN and range queries.

The performance of these two index structures are evaluated in terms of index construction and update time, query response time, retrieval efficiency and distance computations during index construction/update and querying. The query performance of these two index structures are also compared with Sequential Scan. For the evaluation of the retrieval efficiency ANMRR metric [4] that is the evaluation criteria for MPEG-7 visual descriptors and precision/recall scores are used. From the experimental results, it can be stated that BitMatrix along with OWA operator improves the query performance of the content-based retrieval system in terms of both query response time and retrieval efficiency.

The rest of the thesis is organized as follows: Chapter 2 explains MPEG-7 standard in detail and overviews content-based video retrieval. Chapter 3 presents our content-based retrieval system. Chapter 4 discusses Slim-Tree and Chapter 5 discusses BitMatrix used in our system. Chapter 6 gives the implementation details of our system. The performance experiments of the content-based retrieval system are given and the results are discussed in Chapter 7. Finally, Chapter 8 provides conclusions and gives future directions.

CHAPTER 2

BACKGROUND

The organization of this chapter is as follows: The first section gives an overview on MPEG-7 [5] and visual descriptors introduced in MPEG-7. The second section overviews low-level visual features of video, similarity measurement of video data, video indexing and querying concepts in content-based video retrieval systems and discusses some of the content-based video retrieval systems.

2.1 MPEG-7

In this section, a brief discussion on MPEG-7 standard and the visual descriptors introduced by the standard is given.

2.1.1 Introduction

MPEG-7 [5], formally named as Multimedia Content Description Interface, is introduced as an ISO/IEC standard by MPEG (Moving Picture Experts Groups) for representing the audio-visual (AV) content. The scope of MPEG-7 is different from the prior standards (MPEG-1, MPEG-2, and MPEG-4) which focus on the coding of the AV content. MPEG-7 bases on the description of multimedia content and does not standardize the way to obtain these descriptions or to use them, but only standardize the descriptions and the way of structuring them [5].

One of the goals of MPEG-7 is content-based retrieval of AV content. To achieve this goal, MPEG-7 provides tools to describe the AV content. One of these description tools is the MPEG-7 Visual *Descriptors (Ds)* which describe the low-level features such as color, texture, shape and motion of the visual content. MPEG-7 also defines *Description Schemes (DSs)* which specifies the available descriptors for the specified description and the relations between the stated descriptors and/or between other *DSs*. In addition to Ds and DSs, MPEG-7 provides the *Description Definition Language (DDL)*, which enables users

to create their own Description Schemes and Descriptors. DDL contains the syntactic rules to define and combine Description Schemes and Descriptors.

2.1.2 Scope of MPEG-7

Searching, filtering and browsing of the AV content are the candidate application areas of MPEG-7 standard. As shown in Figure 2.1, the scope of MPEG-7 standard is to define the representation of the AV content description. Briefly, any issue that tends to be application-dependent is outside the scope of the standard such as feature extraction and search.

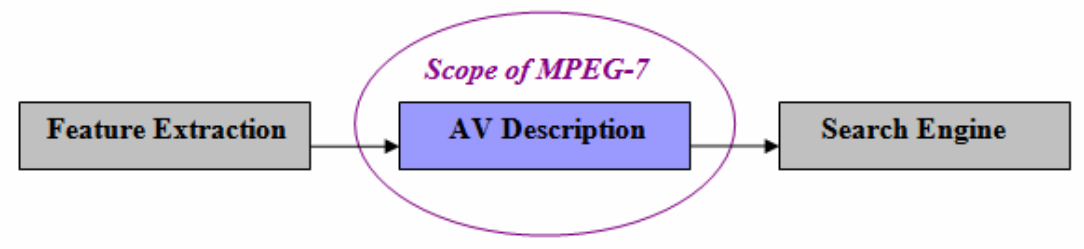

Figure 2.1 Scope of MPEG-7 Standard

2.1.3 MPEG-7 Visual Descriptors

MPEG-7 visual descriptors provide a standard description of visual content such as image and video in searching, browsing and filtering applications. In the following paragraphs, the basic MPEG-7 visual descriptors are described briefly.

2.1.3.1 Color Descriptors

There are seven color descriptors defined in MPEG-7 standard.

- *Color Space:* This descriptor represents the color space that is used in a specific color descriptor (Color Layout, Scalable Color etc.). Color spaces supported by the MPEG-7 standard are *RGB*, *YCbCr*, *HSV*, *HMMD*, *Monochrome* and *Linear transformation matrix with reference to RGB* [4][8].

- *Color Quantization:* This descriptor specifies the quantization of the given color space. Both *Color Quantization* and *Color Space* descriptors are used in conjunction with the other color descriptors specified in the standard.

- *Dominant Color:* This descriptor is an effective and compact descriptor which specifies the representative color(s) (up to 8) in a specified region or in a whole image/frame with the percentage of each representative color in the defined color space and a spatial coherency value. The compactness of the descriptor makes the descriptor a good candidate for content-based indexing.

- *Color Layout:* This descriptor provides information about the spatial distribution of color in a region/image/frame of interest. Discrete Cosine Transform (DCT) is applied on a 2-D array of local representative colors in Y or Cb or Cr color space [4] to represent the spatial color distribution in the frequency domain. The descriptor is compact as the dominant color descriptor; hence, it is easy to index. This descriptor is an efficient descriptor for image-to-image and video clip-to-video clip matching [4].

- *Scalable Color:* This descriptor is a color histogram in HSV color space, encoded using Haar transform [8].

- *Color Structure:* This descriptor captures both color content (similar to a color histogram) and information about the structure of this content. The main functionality of this descriptor is image-to-image matching.

- *Group of Frames/Pictures (GoF/GoP) Color:* This descriptor extends the scalable color descriptor which is defined for a still image to a set of images or frames in a video segment.

2.1.3.2 Texture Descriptors

There are three texture descriptors defined in MPEG-7 standard: Texture Browsing, Homogenous Texture and Edge Histogram.

- *Texture Browsing:* This descriptor provides information about the regularity, directionality and coarseness of the texture of an image/frame. Texture Browsing is a compact descriptor. Since the texture may have more than one dominant

direction, only two dominant directions and related coarsenesses are allowed by the descriptor in the standard.

This descriptor is useful for browsing applications and also can be used for fast and accurate image retrieval in conjuction with Homogenous Texture Descriptor (HTD) [8]. In the context of similarity retrieval, this descriptor can be first used to get candidate images with similar textures, then HTD is used to precise the similarity match among those candidate images. However, it is not recommended by the MPEG-7 standard to use this descriptor in image's texture similarity matching alone because of its retrieval performance. In addition, it is recommended in [4] that the descriptor is better to be used on homogeneously textured images (e.g. textile/fabric patterns).

- *Homogenous Texture:* This descriptor characterizes the region texture in frequency domain by using the mean energy and the energy deviation from a set of frequency channels [4]. This descriptor is used in image/video retrieval. It is recommended in [4] that the descriptor is better to be used on homogeneously textured images (e.g. pattern on fabrics, aerial imagery, Internet images, and trademark images).

- *Edge Histogram:* This descriptor describes the spatial distribution of the edges in an image/frame. Edges in the edge histogram are categorized as vertical, horizontal, 45° diagonal, 135° diagonal and isotropic. This descriptor is useful for image-to-image matching. Since the computation of the edge histogram descriptor is easy and straightforward, it is a good choice for image/video retrieval based on texture. It is recommended in [4] that the descriptor is better to be used on non-homogeneous regions (e.g. natural images, sketch images, clip art images). The performance of the descriptor can be enhanced by using this descriptor in conjunction with other visual features, such as color and shape.

2.1.3.3 Shape Descriptors

There are three shape descriptors defined in MPEG-7 standard: Contour Shape, Region Shape and 3-D Shape.

- *Contour Shape:* This descriptor captures the shape properties of the contour of a given object by using Curvature Scale Space (CSS) respresentation of the contour.

Objects for which characteristic shape features are contained in the contour are described efficiently by this descriptor [9]. The descriptor is very efficient in applications where high variability in the shape is expected, due to, e.g., deformations in the object (rigid or non-rigid) or perspective deformations [9].

- *Region Shape:* This descriptor captures the pixel distribution within a region. Hence, this descriptor considers all pixels in the shape, not only the contour of the object as the contour shape descriptor does. This descriptor can describe complex objects consisting of multiple disconnected regions as well as simple objects with or without holes [9]. The descriptor performs well for shapes where region-based similarity is important (e.g. complex shapes that consist of several disjoint regions) [4].

- *3-D Shape:* This descriptor expresses the geometrical attributes of the 3-D surfaces of objects [4]. Hence, it is used for retrieval or browsing of 3-D objects based on shape.

2.1.3.4 Motion Descriptors

There are four motion descriptors defined in MPEG-7 standard: Motion Activity, Motion Trajectory, Camera Motion, and Parametric Motion.

- *Motion Activity:* A video sequence is perceived by human as being slow, fast etc in terms of motion. For example, “a goal in a football match” is a fast video sequence, where “news reader” video sequence is perceived as a slow one. This descriptor [10] captures this intuitive notion of ‘intensity of action’ or ‘pace of action’ in a video segment [4], and it is measured by using motion vectors of a video sequence. This descriptor consists of the following attributes [10].

    - *Intensity of Activity* – This part of the descriptor is expressed by a 3-bit integer in the range, where “1” indicates low activity and “5” indicates high activity [5][10]. Intensity is defined as the variance of motion vector magnitudes over all frames in the video sequence [11].

    - *Direction of Activity* – This part of the descriptor expresses the dominant direction of the video sequence, if exists. It is expressed by a 3-bit integer

which indicates one of the eight equally spaced angles between 0 and 360 degrees [4].

- *Spatial Distribution of Activity* - This part of the descriptor expresses whether the activity is distributed across many regions or restricted to one region in the sequence [4]. For example, "a busy street" video sequence will have many small active regions, where "news reader" sequence will have only one active region.

- *Temporal Distribution of Activity* - This part of the descriptor expresses the variation of the level of the activity over time [4]. It is actually a motion activity histogram with respect to time. Each value in the histogram indicates the relative percentage of the specified level of activity with respect to whole video sequence.

The descriptor can be used to retrieve the video shots that are similar to the given query video shot in terms of motion activity which is surely the supposed action. As a result, in the result set of video shots, there can be video shots which are similar in motion attributes but not in other characteristics (i.e. color, shape, texture). Hence, motion descriptors must be used in combination with the other characteristics, such as color, texture and shape to get more accurate retrieval results.

- *Motion Trajectory:* This descriptor describes the position of moving objects with respect to time, where the object is defined as the key-point (usually the center of mass) of a moving region whose trajectory is important for the specific application [4]. The descriptor assumes that the spatio-temporal regions are already extracted from the related video sequence. For example, this can be achieved by using alpha-channels in MPEG-4 format. However, for the other video formats, a segmentation process has to be done. Once the regions are extracted from the video content, the extraction of this feature is quite easy and fast. This descriptor can be used in similarity matching of video segments or high-level queries such as "retrieve objects passing near a given area" or "retrieve objects moving faster than a given speed" [4].

- *Parametric Motion:* This descriptor describes the 2-D geometric transformation of a region that the descriptor is associated with over a specified time interval. 2-D geometric transformation models defined in this descriptor are translational,

rotation, affine, planar perspective and parabolic [11]. The descriptor enables to retrieve video objects of similar motions undergoing rotations or deformations that are not captured by motion trajectory descriptor [4]. Although this descriptor is a compact one, it is the most computationally complex motion descriptor of MPEG-7 standard.

- *Camera Motion:* This descriptor describes the 3-D camera motion parameters that can be obtained from the capture devices and the descriptor consists of the types of camera motions present in a given video sequence. By using this descriptor, assumptions can be made on the correlation between the high-level characteristics of the video sequence content and the motion of the camera [4].

2.2 Content-Based Video Retrieval

In Figure 2.2, a typical CBVR system that indexes video data based on its low-level features is represented.

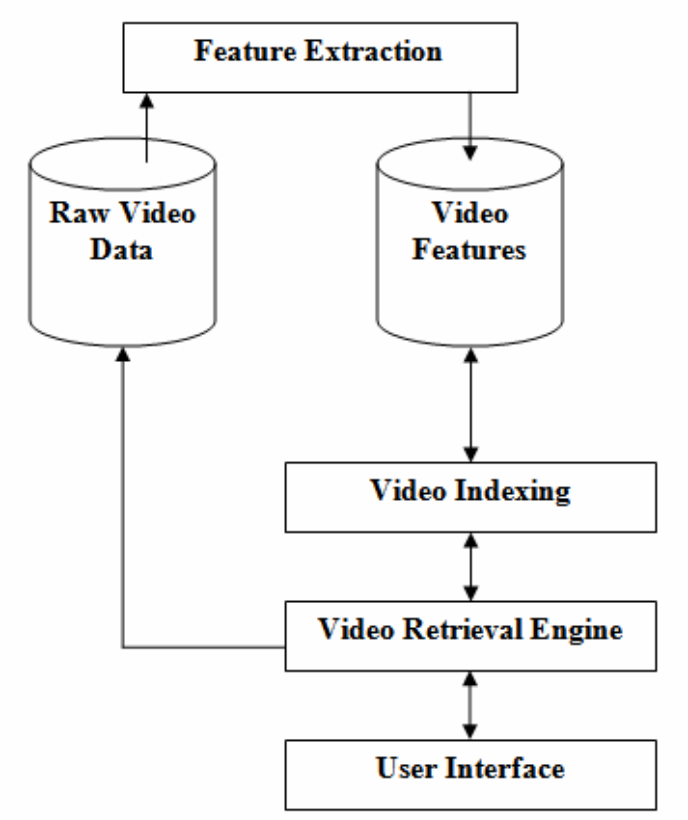

Figure 2.2 Typical Content-Based Video Retrieval System

2.2.1 Video Database

The raw video database contains video shots for the purpose of visual display. Unlike traditional databases, a preprocessing step (i.e. feature extraction) is required for querying the raw video database, because of the nature of video data. The video features database stores needed visual features extracted from videos to support content-based video retrieval.

2.2.2 Low-level Features of Video

There are mainly four low-level features that may be used in content-based video retrieval: color, texture, shape and motion.

2.2.2.1 Color

Color is an important visual attribute for human vision system [4] and independent of related region's size and orientation [12]. Thus, color feature is one of the most widely used low-level feature in content-based video retrieval.

2.2.2.2 Texture

Texture has emerged as an important visual primitive for searching and browsing through large collections of similar looking patterns [4]. Thus, texture feature is also a widely used low-level feature and facilitates content-based video retrieval.

2.2.2.3 Shape

Shape feature is very powerful when used in similarity search and retrieval, since the shape of objects is usually strogly linked to object functionality and identity [4]. Humans can recognize characteristics objects solely from their shapes, and this property distinguishes shape from other low-level features such as color and texture [4].

2.2.2.4 Motion

Motion feature is the key low-level feature in video indexing, since it provides the easiest access to the temporal dimension of video [4]. Motion feature improves the performance of content-based video retrieval when used with the other low-level features such as color, texture and shape [4].

2.2.3 Similarity Measurement

Similarity between a query video and videos stored in video database is measured by computing the distance between the low-level visual features of the related videos. The similarity/distance measures are mathematical definitions of the similarity and the distance measure(s) used in similarity matching of videos affect the retrieval performance. Thus, the choice of using which distance measure is important. A distance measurement need not to be metric. However, metric distances are decided to be used in this study. A metric distance measure must satisfy the following properties [39]:

1. $D(p, q) \geq 0$ ($D(p, q) = 0$ iff $p = q$),
2. $D(p, q) = D(q, p)$, and
3. $D(p, z) \leq D(p, q) + D(q, z)$

In metric spaces, what is actually measured is the distance between feature values [17][18], so the distance function returns a dissimilarity value between any two objects, where high distance corresponds to low similarity and low distance corresponds to high similarity.

In the literature, many distance measures have been proposed and some of most commonly use distances are listed below [12][38]:

- *Minkowski-form Distance*: This distance is defined as:

$$Dp(x,y) = \left[\sum_{i=0}^{d-1} \left| x(i) - y(i) \right|^{p} \right]^{\frac{1}{p}} \quad \textbf{(1)}$$

where x and y are feature vectors and d is feature dimension. This distance measure is appropriate when each dimension of video feature vector is independent of each other and is of equal importance.

- *Weighted Minkowski-form Distance*: This distance is defined as:

$$Dp(x,y) = \left[\sum_{i=0}^{d-1} w_i \left| x(i) - y(i) \right|^p \right]^{\frac{1}{p}} \quad \textbf{(2)}$$

With weighted version of Minkowski-form distance, each dimension of video feature vector can be weighted by assigning different non-negative weights to each dimension.

- *Euclidean Distance*: When p = 2 in the definition of Minkowski-form distance, it is called as Euclidean distance and defined as:

$$D_2(x,y) = \left[\sum_{i=0}^{d-1} \left| x(i) - y(i) \right|^2 \right]^{\frac{1}{2}} \quad \textbf{(3)}$$

- *Weighted Euclidean Distance*: When p = 2 in the definition of weighted Minkowski-form distance, it is called as weighted Euclidean distance and defined as:

$$D_2(x,y) = \left[\sum_{i=0}^{d-1} w_i \left| x(i) - y(i) \right|^2 \right]^{\frac{1}{2}} \quad \textbf{(4)}$$

- *Manhattan Distance*: When p = 1 in the definition of Minkowski-form distance, it is called as Manhattan distance (i.e. city-block distance) and defined as:

$$D_1(x,y) = \left[\sum_{i=0}^{d-1} \left| x(i) - y(i) \right| \right] \quad \textbf{(5)}$$

- *Chebychev Distance*: When p = ∞ in the definition of Minkowski-form distance, it is called as Chebychev distance and defined as:

$$D\infty(x,y) = \max_{i=0}^{d-1} \left| x(i) - y(i) \right| \quad \textbf{(6)}$$

- *Mahalanobis Distance*: This distance is defined as:

$$D(x,y) = \left| \det C \right|^{\frac{1}{d}} (x-y)^T C^{-1} (x-y) \quad \textbf{(7)}$$

where C is the covariance matrix of the feature vectors. This distance can be used when each dimension of video feature vector is dependent to each other and have different weights.

2.2.4 Video Indexing

Due to the complexity inherent to video data, when the number of dimensions of feature vectors and/or the number of video objects to be indexed increase, sequential scan of the feature vectors becomes inefficient in terms of response times of similarity queries. Many techniques and systems have been proposed in the context of multi-dimensional indexing that can be used to provide efficient content-based video retrieval. The common goal of these multi-dimensional indexing methods is to divide the search space in a set of ranges and prune some of them at search time [1].

Vector-space methods index the feature vectors of multi-dimensional data. These index structures are based on a tree data structure with the data nodes in the leaves of the tree and a cluster hierarchy is built on top [1]. There are mainly two data partitioning strategy in vector-space methods [1]:

- *Data Partitioning*: This partitioning strategy uses *minimum bounding rectangles (MBR)* such as R-tree, R*-tree, X-tree, *bounding spheres* such as SS−tree, *MBR and bounding spheres* such as SR−Tree, *generic minimum bounding regions (hyper rectangle, cube, sphere)* such as TV −tree [16].

- *Space Partitioning*: This partitioning strategy partitions the space rather than the data. kDB−tree, Hybrid−tree and SH−tree are examples of this type of index structures that use space partitioning [16].

Metric-space methods such as M-Tree [13] and Slim-Tree [3] index the multi-dimensional data based on the relative distances between indexed objects, not based on the feature

vectors of the indexed objects as vector-space methods do. Metric-space methods are also based on a tree data structure with the data nodes in the leaves of the tree and a cluster hierarchy is built on top [1].

Single-Dimensional Mapping techniques such as Pyramid-Technique [19] and KPYR [22] map the points in the multi-dimensional space to the single-dimensional values for which efficient techniques exist [1].

Data Approximation Structures such as VA-File [21], OVA-File [20] and BitMatrix [1][2] make the assumption that a sequential scan is inevitable and construct a vector of approximations (VA-file), significantly smaller than the original data [1]. In the first step, the approximations are pruned based on their minimum/maximum distance to the query point and then for the remaining approximations, corresponding exact data points are analyzed [1].

2.2.4.1 Overview of Some Multi-dimensional Index Structures

R-Tree [23] is a vector-space indexing method and partitions the data space by using Minimum Bounding Rectangles (MBRs). This index structure has been originally designed for spatial databases [16][23]. R-Tree is a tree structure which includes two types of nodes: internal nodes that cover all MBRs in the lower level of the tree and leaf nodes that contain MBRs of indexed objects and a pointer to the indexed object.

R-Tree* [24] is an extension of the R-Tree based on a careful study of the R-Tree algorithms under various data distributions [16]. The objectives of R*-Tree are to minimize overlap between nodes, the volume covered by internal nodes, and to maximize the storage utilization [16].

R^+-Tree [25] is an overlap-free variant of R-Tree [16]. The index structure of R^+-Tree is as R-Tree's. However, R^+-Tree introduces *force-split* strategy that propagates the split of nodes downward except leaf nodes [25]. The basic idea is to reduce overlap between nodes, but propagating split downwards may degrade the storage utilization.

R-Tree, R*-Tree and R^+-Tree are originally designed for two-dimensional spatial objects, but also have been used for multi-dimensional objects [16]. The basic problem of these structures is the increasing overlap between nodes when used on multi-dimensional objects. Thus, another index structure, X-Tree [26], based on R*-Tree is introduced that is originally

designed for multi-dimensional objects. X-Tree introduces a new split algorithm that minimize overlaps between nodes and *supernode* concept that allows the extension of an internal node of the tree to avoid split. The basic idea of minimizing overlap and supernode concept is to keep the tree as hierarchical as possible and to avoid splits in the tree that would result higher overlap [26].

As the dimensionality of the feature vectors of data objects increase, the performance of multi-dimensional indexing methods such as R-Tree, R*-Tree, R^+-Tree and X-Tree degrades. The specified methods are outperformed by simple sequential scan if the number of dimensions exceeds a certain value. To tackle this issue called as *the curse of dimensionality*, *Vector Approximation (VA-File)* [21] method is proposed. The objective of this method is to make the unavoidable sequential scan as fast as possible by using vector approximations to compress the vector data. During k-NN similarity queries, entire VA-File is scanned and vast majority of the vector data objects are filtered out based on the vector approximations. Hence, the search space is reduced. After this filtering step, the remaining vector data points are visited sequentially in increasing order of the distance to the query vector data point.

A variant of VA-File, namely *OVA-File* [20] index structure aims an efficient solution for content-based video indexing. It is basically based on VA-File. However, OVA-File is a hierarchical index structure. It partitions the vector approximation file into slices such that only a part of these slices are processed during k-NN queries. OVA-File is a dynamic and balanced index structure. It handles the insertion of new vector approximations into file efficiently.

In VA-File method since all approximation file is scanned sequentially, query time is almost constant. The query time of OVA-File varies, since the size of the visited parts of an OVA-File is different for different queries. Also in OVA-File, there is a tradeoff between the query result quality and the query response time.

M-Tree [13] is a paged, dynamic and balanced metric tree. It does not need periodic reorganization of the tree structure after some updates (i.e. insertions and/or deletions) on the tree, since it organizes its structure during each insert or delete operation. It is proposed to organize and search large data sets from a generic “metric space”, i.e. where object proximity is only defined by a distance function satisfying the positivity, symmetry, and triangle inequality postulates [13]. Since the design of M-tree is inspired by both principles

of metric trees and database access methods, performance optimization concerns both CPU (distance computations) and I/O costs [13].

Pyramid-Technique [19] is an indexing method that maps the d-dimensional data space into a 1-dimensional space and then uses B^+-Tree index structure to index the 1-dimensional data space.

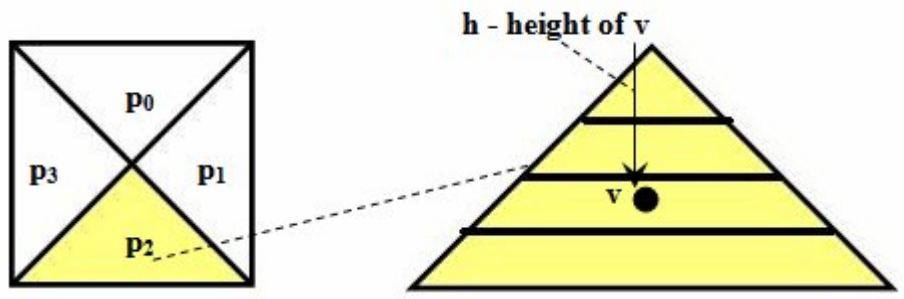

Figure 2.3 Sample 2-dimensional Data Space

In Pyramid-Technique, d-dimensional space is partitioned into 2d pyramids. In Figure 2.3, a sample 2-dimensional space and its partitioning into pyramids are given: 2-dimensional space is partitioned into 4 pyramids. After data space is partitioned into pyramids, pyramid value of each d-dimensional point, v is calculated as:

$$pv(v) = (i + h)$$

where i is the identifier of the pyramid where v is located and h is the height of v in its pyramid. The pyramid values of d-dimensional points are then indexed by using one of the one-dimensional index structures such as B+-Tree.

Although the Pyramid-Technique performs well for hypercube shaped range queries, for very skewed queries or queries specifying only one attribute, the Pyramid-Technique performs worse than the sequential scan [19]. The performance of Pyramid-Technique is sensitive to the positions of the query hypercube; it is less effective for clustered data sets; and it is inferior for partial range queries [20]. Thus, using sequential scan method on the data space may be more effective in the stated cases.

KPYR [22] is an index structure which is basically proposed for video data. It uses a well known clustering technique *k-means* to index multi-dimensional video data. The basic idea is to cluster the data first into *k* clusters and then using Pyramid-Technique on each cluster

that is transformed into a hypercube to apply Pyramid-Technique. In Figure 2.4, geometrical representation of KPYR is given.

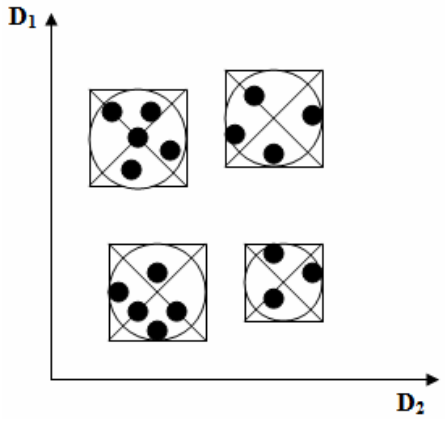

Figure 2.4 Geometrical Representation of KPYR

As shown in Figure 2.4, 2-dimensional points are clustered into four cluster to apply Pyramid-Technique on each cluster. The performance tests show that KPYR outperforms Pyramid Technique and sequential scan for range queries under different workloads (i.e. different data set sizes, dimensionality and selectivity) [22]. Further information about performance tests can be found in [22].

STRG-Index [45] is a graph-based index structure and aims at representing spatial features of video objects and temporal relationships between video objects. STRG-Index expresses these spatio-temporal characteristics of video as Spatio-Temporal Region Graph (STRG). An STRG represents the spatial relations between adjacent video objects and temporal relationships between the corresponding video objects in two consecutive frames of video. This index structure constructs foreground and background STRGs for a given video and indexes these foreground and background STRGs using a tree structure where root nodes contain background STRGs, leaf nodes contain foreground STRGs and internal nodes contain representative foreground STRG for clustered foreground STRGs.

2.2.5 Video Querying

There are two principal ways for the representation of queries for video data, namely *Query-By-Example* and *Query-By-Subject* [36]:

- *Query-By-Example (QBE)*: This method allows an intuitive representation of queries. A query may be represented by drawing a rough sketch, a rough painting using colors or an motion trajectory of an object. Such representations for query conditions for video data are better than keywords, since it is difficult to express slight differences in color, texture shape and motion with keywords [36].

- *Query-By-Subject (QBS)*: This method allows specifying a subjective description of a query and in such cases knowledge is required to capture the semantic contents of video data to interpret the query [36]. Extraction of semantic content from the raw video data is proven be a difficult task. The simplest way of semantic content management is to annotate video data with text [36]. However, since this annotation with text requires human-interaction with the retrieval system, it is not practical for large video databases. Another way is using a knowledge-based system and by using the knowledge database interpreting video queries on particular subject(s).

2.2.6 Content-Based Video Retrieval Systems

In the literature, many content-based video retrieval systems are presented. In this section, we discuss some of these systems.

2.2.6.1 VIRS

VIRS [41] is a video/image retrieval system based on MPEG-7 visual descriptors. The system uses Dominant Color, Color Layout and Color Structure, Homogeneous Texture, Texture Browsing, Edge Histogram, Motion Activity, Motion Trajectory, Camera Motion and Region Shape visual descriptors to describe the image/video content. The system provides two types of query method [41]: one is Query-By-Example (QBE) method which searches the similar images/videos with a sample, and the other is Query-By-Draw (QBD) method which retrieves by rough sketch drawing. The system also supports giving various weights to each feature used in querying.

2.2.6.2 BilVMS

BilVMS [42] is a state-of-art video management system that indexes the video data based on MPEG-7 visual descriptors. The system is capable of temporally segmenting video into shots, as well as obtaining a semantically meaningful group of shots, i.e. scenes [42]. BilVMS uses Dominant Color, Color Layout, Scalable Color, Color Structure, Homogeneous Texture, Edge Histogram, Camera Motion and Motion Activity visual descriptors to index video. The system uses the keyframes of video shots to represent the still image visual features (i.e. visual features other than motion descriptors). In this study, a number of semantic classes are defined and keyframes of video shots are assigned into those semantic classes according to their MPEG-7 color and texture features [42]. The system also includes videotext detection/recognition and face detection capabilities.

2.2.6.3 MUVIS

MUVIS [37] provides a framework for content-based indexing and querying multimedia collections such as image, audio and video data. The system uses low-level features of multimedia data for indexing and merges several features by weighted interpolation during queries. Keyframes of video clips are used to represent the still image features of the video clips.

2.2.6.4 WebSEEk

WebSEEk [40] is a cataloguing tool that allows content-based image and video querying on the Web. WebSEEk uses several autonomous agents that collect, index and assign image/video objects into defined subject classes. The system provides querying the indexed images/videos by using keywords or visual content.

2.2.6.5 QBIC: Query By Image and Video Content

QBIC [44] is a system that provides content-based image and video retrieval. In QBIC, firstly videos are broken into video shots and representative frames are extracted for each video shot. These representative frames are used to describe the still image features of the video shots. Further processing of video shots generates

motion objects, for example a car moving across the screen [44]. Queries are allowed on objects (“Find images with a red, round object”), scenes (“Find images that have approximately 30-percent red and contain a blue textured object”), shots (“Find all shots panning from left to right”), or any combination [44]. In QBIC, the similarity measurement between the features of images/videos is done by using distance functions.

2.2.6.6 Virage Video Engine

Virage Video Engine [43] provides a framework and tools for content-based video retrieval. The engine is a flexible platform-independent architecture which provides support for processing multiple synchronized data streams like image sequences, audio and closed captions and the architecture of the system allows for multi-modal indexing and retrieval of video [43].

CHAPTER 3

THE CONTENT-BASED RETRIEVAL SYSTEM

In this chapter, our content-based retrieval system that adapts BitMatrix and utilizes Slim-Tree index structures for efficiently retrieving images and videos based on low-level descriptors is presented. The parts we focus during the content-based retrieval system development are the detection of video shots and the selection of keyframes for the detected video shots, low-level feature extraction and storage of images and video shots, indexing and querying. These parts of the system are explained in detail in the following sections.

3.1 Overview of the Content-Based Retrieval (CBR) System

In Figure 3.1, the block diagram of our content-based retrieval system is represented.

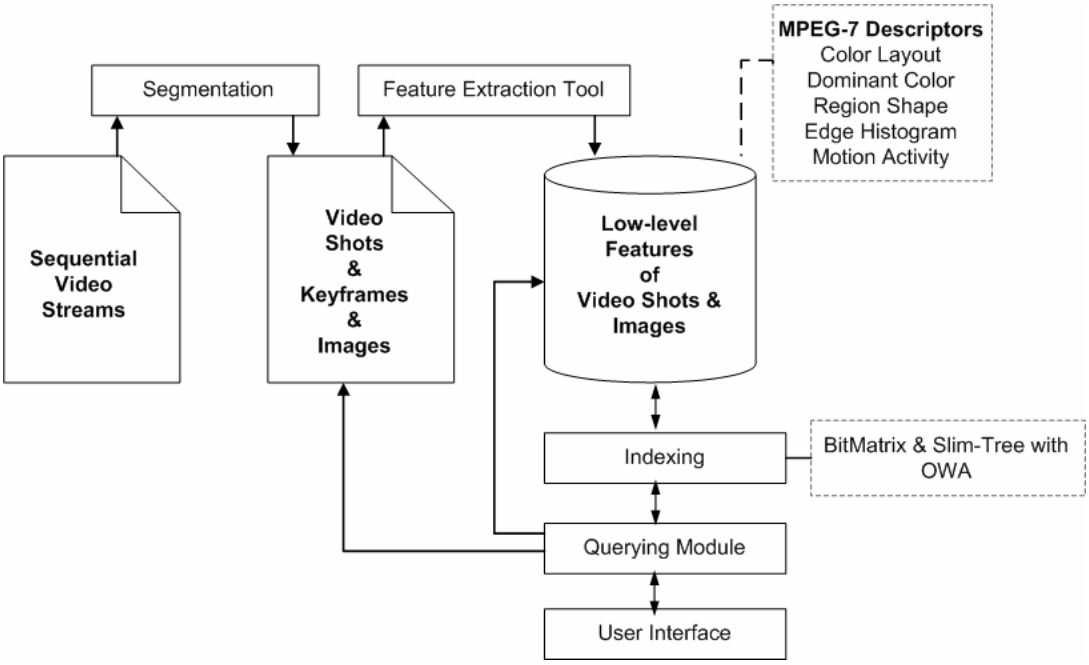

Figure 3.1 Block Diagram of the CBR System

As seen from Figure 3.1, sequential video streams are broken into manageable units, namely video shots and the keyframes of the video shots are extracted. The resulting video shots, keyframes and also images are put into a pre-defined location on the file system. After this step, the low-level features of video shots and images are extracted. The low-level features extracted for video shots are Color Layout (CL), Dominant Color (DC), Edge Histogram (EH), Region Shape (RS) and Motion Activity (MA). The still image features such as Color Layout (CL), Dominant Color (DC), Edge Histogram (EH) and Region Shape (RS) of video shots are extracted by using the related keyframes of the video shots. In our system, one keyframe is used for each video shot to represent the still image features of the video shots. The low-level features extracted for images are same as for video shots except Motion Activity (MA). The generated low-level features of video shots and images are stored in a native XML database. The video shots and images are indexed based on their low-level features stored in the XML database by two different index structures, namely Slim-Tree and BitMatrix. The retrieval engine uses these two index structures to retrieve video shots and images based on their content.

Our content-based retrieval system is capable of indexing and retrieving both images and video shots. However, our main focus is the content-based video retrieval. Thus, we explain the parts of our system from this perspective. The parts of our content-based retrieval system are discussed in detail, in the following sections.

3.2 Video Shot Detection and Keyframe Extraction

Before feature extraction process, firstly sequential video streams are broken into manageable units, namely *video shots*, whose content is similar as a whole and the keyframes of the video shots are detected. This process is briefly explained with Figure 3.2.

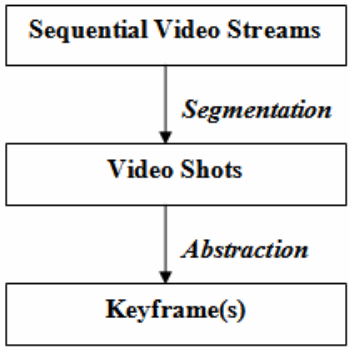

Figure 3.2 Parsing Video Content for Indexing

The keyframes of video shots are used to extract the still image features of the video shots such as Color Layout (CL), Dominant Color (DC), Edge Histogram (EH) and Region Shape (RS). To represent the still image features of a video shot, one or more keyframe can be used. In this study, we use one keyframe for each video shot. Video shot detection and keyframe extraction processes are done offline in the system. The detected video shots and the extracted keyframes of the video shots are the input for feature extraction process of the system.

3.3 Feature Extraction

In [33], the visual content descriptors, which are extracted with MPEG-7 Descriptors, are analyzed from the statistical point of view. The main results show that the best descriptors for combination are Color Layout (CL), Dominant Color (DC), Edge Histogram (EH) and Texture Browsing (TB) and the others are highly dependent on these. In this study, we use as color descriptor Color Layout (CL) and Dominant Color (DC); as texture descriptor Edge Histogram (EH); as shape descriptor Region Shape (RS); and as motion descriptor (only for video shots) Motion Activity (MA) defined in MPEG-7 Standard. These descriptors are explained in detail, in Section 2.1.3. In our content-based retrieval system, after video shots and keyframes are detected, visual features of video shots such as CL, DC, EH, RS and MA are extracted and stored in XML format.

3.4 Raw Data and Feature Storage

The video shots and their related keyframes are stored in a pre-defined location on the file system for further processing (i.e. low-level feature extraction). The generated XML files that contain visual features of the video shots are stored in a native XML database. The storage of extracted features into the XML database are done offline.

3.5 Similarity Measurement

In metric space structures, what is actually measured is the distance between feature values [17][18]. The distance function that is used to measure similarity returns a dissimilarity value between any two objects and high distance corresponds to low similarity, where low distance corresponds to high similarity. Hence, in our system, similarity measurement between any two video shots is carried out by using distance functions. In the following two

subsections, distance functions used for similarity measurement, OWA operator and its usage in the system are discussed, respectively.

3.5.1 Distance Function

Commonly used distance functions are explained in Section 2.2.3. In this section, distance functions used in the system are presented. Since the content of visual descriptors (i.e. CL, DC, RS, EH and MA) used in the system is explained in Section 6.2, the meaning of parameters used in distance functions are not specified in this section.

In this study, we use the following distance function for the Color Layout (CL) feature:

$$D_{CL} = \sqrt{\sum_{i=1}^{6}\left(YCoeff[i] - YCoeff'[i]\right)^2} + \sqrt{\sum_{i=1}^{3}\left(CbCoeff[i] - CbCoeff'[i]\right)^2} + \sqrt{\sum_{i=1}^{3}\left(CrCoeff[i] - CrCoeff'[i]\right)^2} \quad \textbf{(8)}$$

Distance function for the Edge Histogram (EH) feature:

$$D_{EH} = \sqrt{\sum_{i=1}^{n}\left(Bincounts[i] - Bincounts'[i]\right)^2} \quad \textbf{(9)}$$

where n = 80.

Distance function for the Region Shape (RS) feature:

$$D_{RS} = \sqrt{\sum_{i=1}^{n}\left(MagntOfART[i] - MagntOfART'[i]\right)^2} \quad \textbf{(10)}$$

where n = 35.

Distance function for the Motion Activity (MA) feature:

$$D_{MA} = \sqrt{(Nsr - Nsr')^2 + (Nmr - Nmr')^2 + (Nlr - Nlr')^2} + \sqrt{\sum_{i=1}^{5} (tempParam[i] - tempParam'[i])^2} + \sqrt{(paceOfMotion - paceOfMotion')^2} \quad \textbf{(11)}$$

where

paceOfMotion is the motion intensity of video shot,

Nsr, Nmr and Nlr are the spatial distribution parameters, and

tempParam is the motion activity histogram.

Dominant Color (DC) distance function used in this study differs from other features' distance computation. The distance function for DC is a fuzzy distance function which is introduced in [34] and used in [12]. In the computation of DC distance, color similarity is taken into account by applying Single Mode DC Search [34]. Firstly, the system evaluates the differences between the color indexes of any two dominant colors of two video shots in the defined color space which is RGB in this study. If differences are less than a pre-defined threshold value which is 10 in this study, then color similarity is evaluated by using the following function:

$$D_{DC} = \sqrt{\sum_{i=1}^{n} (ColorVal[i] - ColorVal'[i])^2} \quad \textbf{(12)}$$

where n = 3. In the second phase, minimum percentage of the related dominant colors of two video shots is normalized and multiplied by the color similarity and the resulting value is extracted from one to get the final DC similarity of the related dominant colors of two video shots. For all colors defined in first and second video shot, this process is applied and the minimum DC similarity is treated as the overall DC distance between two video shots.

3.5.2 Ordered Weighted Averaging (OWA) Operator

An OWA operator [35] of dimension *n* is a mapping:

$$F : R^n \rightarrow R, \tag{13}$$

which has an associated weighting vector *W*

$$\mathrm{W} = \left[w_1 w_2 ... w_n\right]^{\mathrm{T}} \tag{14}$$

such that

$$\sum_{i=1}^{n} w_i = 1 \tag{15}$$

where $w_i \in [0, 1]$ and where

$$F\left(a_1,...,a_n\right) = \sum_{i=1}^{n} w_i \cdot b_i \tag{16}$$

where b_i is the i^{th} largest element of the collection of the aggregated objects $a_1,\ldots,a_n$. The function value $F(a_1,\ldots,a_n)$ determines the aggregated value of arguments, $a_1,\ldots,a_n$.

To compute the overall distance between two video shots, we compute CL, DC, EH, RS and MA distances and apply normalization to each of them separately so that their range is from '0' (similar) to '1' (dissimilar). After normalization of each feature's distances, we compute the overall distance value from CL, DC, EH, RS and MA distances by using the OWA operator. From the definition of OWA operator, the overall distance is also in [0, 1].

Suppose that (d_1, d_2, d_3, d_4, d_5) are the distance values for color layout, dominant color, region shape, edge histogram and motion activity between any two video shots where $d_1 \leq d_2 \leq d_3 \leq d_4 \leq d_5$. The OWA operator associated to the five nonnegative weights ($w_1,\ldots,w_n$) where $w_i \in [0,1]$ and $w_5 \leq w_4 \leq w_3 \leq w_2 \leq w_1$. It should be noted that the weight w_n is linked to the greatest distance value, d_n, and w_1 is linked to the lowest distance value, d_1, to emphasize similarity between two objects. The final distance value between any two video shots becomes:

$$D = \sum_{i=1}^{n} w_i \cdot d_i \tag{17}$$

where n = 5.

For example, for two video shots O_1 and O_2, we want to compute distance between them, $d(O_1, O_2)$, and assume that, for each feature, CL, DC, EH, RS and MA the normalized distance values are; $d_{DC}(O_1, O_2) = 0.325$, $d_{CL}(O_1, O_2) = 0.570$, $d_{EH}(O_1, O_2) = 0.450$, $d_{RS}(O_1, O_2) = 0.250$, $d_{MA}(O_1, O_2) = 0.32$.

and the OWA weights are;

$w_1 = 0.3$, $w_2 = 0.3$, $w_3 = 0.2$, $w_4 = 0.1$, $w_5 = 0.1$,

so $w_1 + w_2 + w_3 + w_4 + w_5 = 0.3+0.3+0.2+0.1+0.1 = 1$,

then the overall distance is:

$d(O_1, O_2) = F(d_{CL}(O_1, O_2), d_{DC}(O_1, O_2), d_{EH}(O_1, O_2), d_{RS}(O_1, O_2) , d_{MA}(O_1, O_2))$

$= w_1 * d_{RS}(O_1, O_2) + w_2 * d_{MA}(O_1, O_2)$

$+ w_3 * d_{DC}(O_1, O_2) + w_4 * d_{EH}(O_1, O_2) + w_5 * d_{CL}(O_1, O_2)$

$=0.3*0.250+0.3*0.32+0.2*0.325+0.1*0.450+0.1*0.570$

$= 0.338$.

Since there are five low-level features that represent the visual content of a video shot, the system evaluates different distance values for each feature, by using distance functions and combines these values into single value by using OWA operator. In this study, the weight values of OWA operator for the video shots are {0.3, 0.3, 0.2 , 0.1, 0.1} and the weight values of OWA operator for the images are {0.4, 0.3, 0.2, 0.1}.

3.6 Indexing

An ideal content-based video retrieval system shall be scalable to large video collections. Thus, multi-dimensional index structures are needed to provide efficient content-based video retrieval. The multi-dimensional index structures used in the system shall support efficient k-NN and range queries.

For indexing the visual features of video shots, we utilize Slim-Tree [3] known as a dynamic and balanced access structure and adapt BitMatrix [1][2] which is a highly parametrizable structure in our content-based retrieval system. Slim-Tree and BitMatrix are

discussed in detail in Chapter 4 and Chapter 5, respectively. Thus, in this section we explain briefly the construction of Slim-Tree and BitMatrix index structures in the content-based retrieval system.

In our system, Slim-Tree is the index structure utilized to index the low-level features of the video shots. Since Slim-Tree is a metric index structure, it needs distance functions that are used during index construction/update and retrieval to measure the similarity of any two video shots. Thus, the inputs for the construction of Slim-Tree are the low-level feature values (i.e. color layout, dominant color, edge histogram, region shape and motion activity descriptors) of the video shots and the distance functions to be used during index construction. The distance functions provided to Slim-Tree are the ones that are explained in Section 3.5.1.

We use a single Slim-Tree for indexing all the visual features of video shots together. When a distance computation is necessary during index construction/update, our system computes distance values of each visual feature seperately and combines these values into a single distance value by using OWA operator.

In this study, BitMatrix is adapted to video databases for the first time. Unlike Slim-Tree, BitMatrix does not need distance functions during index construction/update, since the index construction/update are only based on the clustering results of low-level visual features of video shots. Thus, the input for the construction of BitMatrix is the low-level feature values (i.e. color layout, dominant color, edge histogram, region shape and motion activity descriptors) of the video shots. We use a single BitMatrix for indexing all the visual features of video shots together, as in Slim-Tree case.

3.7 Querying Module

Our content-based retrieval system uses *Query-By-Example* method. Hence, the system needs low-level visual features of the query video shot for similarity searches. In our system, we use the video shots whose low-level features are already extracted and stored in the XML database as query video shots. Thus, the system fetches the low-level features of the query video shot from the XML database during queries. To retrieve video shots, Slim-Tree and BitMatrix index structures are used by the querying module of the system. Exact match, k-NN and range queries are supported by the system. The stated queries are defined as follows:

Range Query: Given a query object $q \in$ D, where D is the search space (i.e. the domain of indexed features of objects). Range query selects all indexed objects q_j that are within a specified distance (*r(q)*) from the query object, *q*. For example, a range query becomes:

"*Find all video shots which are within 0.3 distance from given query video shot*"

k-NN Query: Given an integer *k* and a query object $q \in$ D, where D is the search space (i.e. the domain of indexed features of objects) and k≥1. k-NN query selects the *k* indexed objects that are closest (i.e. that have the shortest distances) to the query object, *q*. For example, a k-NN query becomes:

"*Find 20 nearest video shots to given query video shot*"

Exact Match Query: Given a query object $q \in$ D, where D is the search space (i.e. the domain of indexed features of objects). Exact match query selects the indexed objects whose distances are zero to the query object, *q*. Thus, an exact match query is actually a range query whose radius (*r(q)*) is zero. An exact match query becomes:

"*Find all video shots which are within 0 distance from given query video shot*"

Querying on Slim-Tree and BitMatrix are discussed in detail in Chapter 4 and Chapter 5, respectively. Thus, in this section we only give the inputs to Slim-Tree and BitMatrix index structures for k-NN and range queries.

For k-NN query on Slim-Tree/BitMatrix, the inputs that shall be provided to the index structure are:

- The number of nearest neighbour objects to retrieve, *k*
- Low-level visual features of the query video shot; color layout, dominant color, region shape, edge histogram and motion activity,
- The distance function for each low-level visual feature, and
- OWA weights used for the aggregation of the distance values of low-level visual features

For range query on Slim-Tree/BitMatrix, the inputs that shall be provided to the index structure are:

- Radius of range query, r
- Low-level visual features of the query video shot; color layout, dominant color, region shape, edge histogram and motion activity,
- The distance function for each low-level visual feature, and
- OWA weights used for the aggregation of the distance values of low-level visual features

CHAPTER 4

SLIM-TREE

In this chapter, the index structure utilized in our content-based retrieval system, namely Slim-Tree is discussed, in detail.

4.1 Introduction

Slim-Tree [3] is a paged, dynamic and balanced metric tree as M-Tree that is mentioned in Section 2.2.4. As stated in [3], the design objective of Slim-Tree is to reduce the overlap between the nodes in a metric tree, since the overlap between nodes affects the performance of index structures. As the other proposed metric trees, Slim-Tree partitions the search space using the relative distances between indexed objects, not the complex features of objects. Minkowski-form distances (Manhattan distance, Euclidean distance etc.) can be used to calculate the relative distances of the objects. Slim-Tree differs from the previous proposed metric trees in the following ways [3]:

- Minimal Spanning Tree (MST) split algorithm that performs faster than other split algorithms without sacrificing search performance is introduced.
- A new algorithm that leads to considerably higher storage utilization is presented to guide the selection of a subtree during an insertion of an object at an internal node.
- Slim-down algorithm is presented to make the metric tree tighter and faster in a post-processing step.

4.2 Structure of Slim-Tree

In Slim-Tree, data is stored in the leaf nodes and a cluster hierarchy is built on top as in other metric trees such as M-Tree [3]. An example view of the Slim-Tree index structure is shown in Figure 4.1 and Figure 4.2.

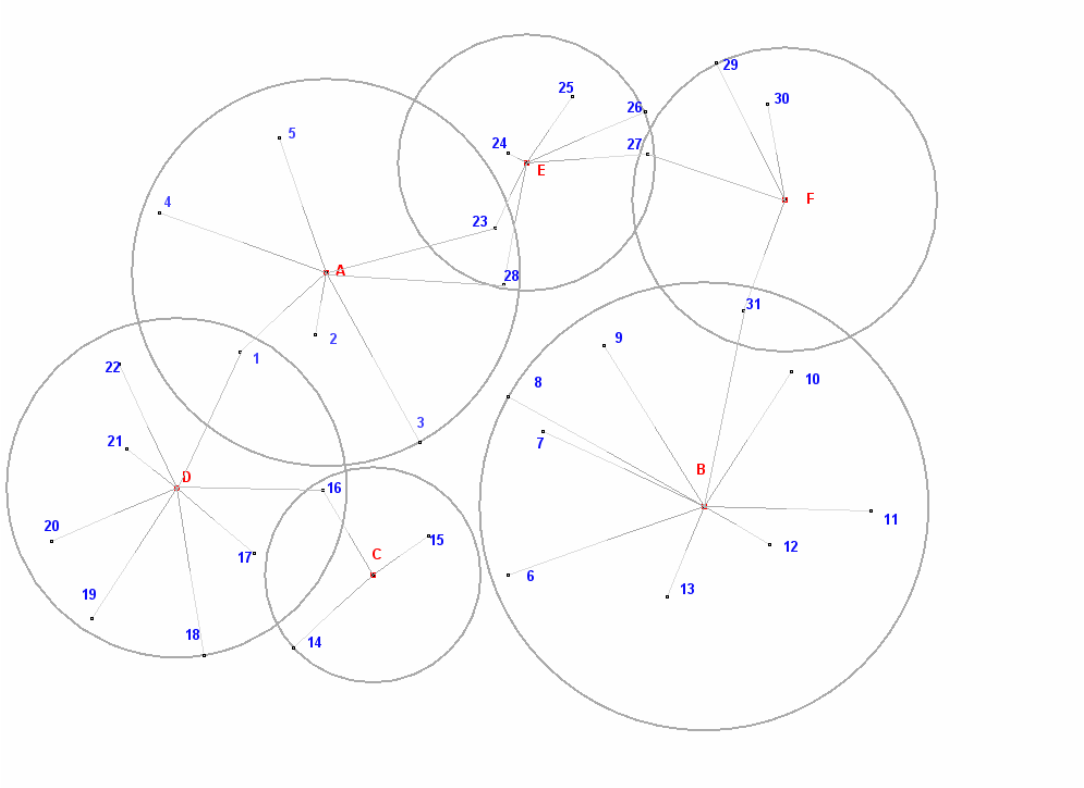

Figure 4.1 Example Data Distribution and Covering Regions

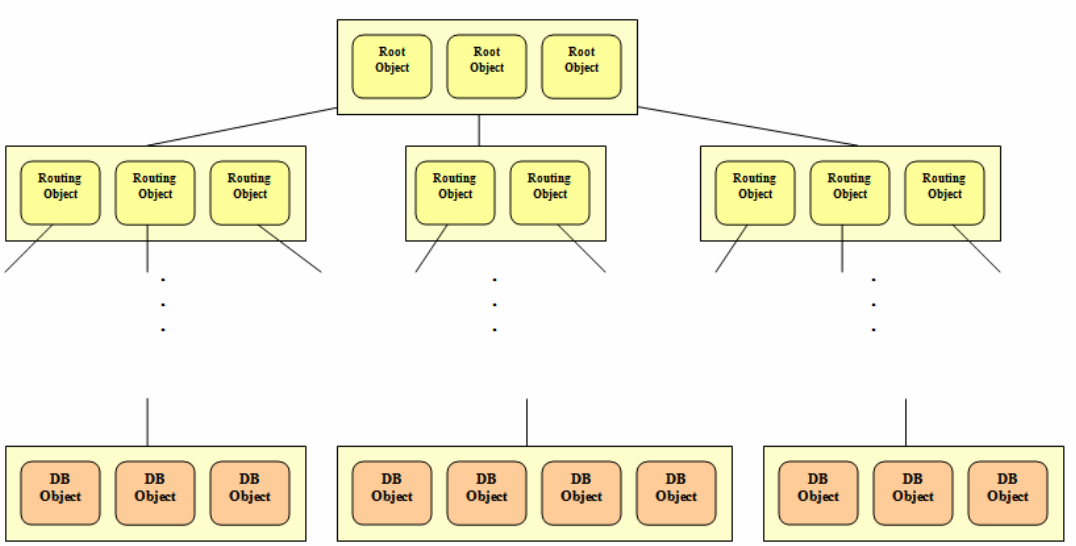

Figure 4.2 General Structure of Slim-Tree

In Figure 4.1, we see thirty one indexed objects with six covering regions and Figure 4.2 shows the general tree structure built for indexed objects.

Slim-Tree has two types of nodes: *internal node* and *leaf node*. A leaf node stores indexed database objects (i.e. ground objects) and an internal node stores routing objects which are also database objects that routing role assigned by a specific split policy.

The information about a routing object is as follows [13]:

- *Or*: (feature value of the) routing object,
- *ptr(T(Or))*: pointer to the root of *T(Or)*, where *T(Or)* is a sub-tree,
- *r(Or)*: covering radius of *Or* (i.e. the maximum of the distances to the child objects),
- *d(Or, P(Or))*: distance of *Or* from its parent, where *P(Or)* is the parent of routing object.

The graphical representation of a routing object of Slim-Tree is given in Figure 4.3:

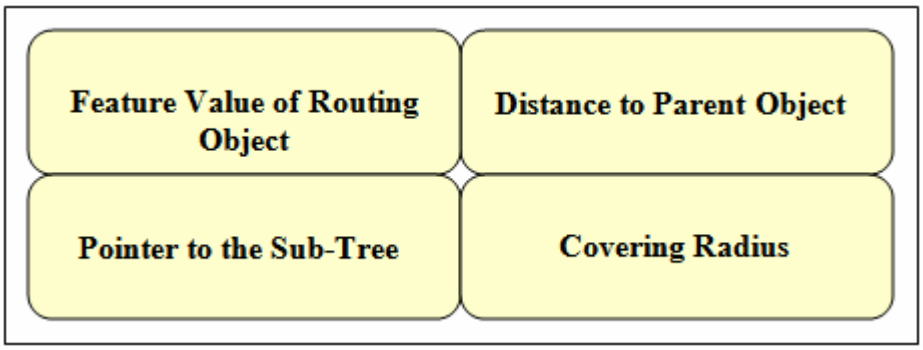

Figure 4.3 Structure of Routing Object in Slim-Tree

The information about a ground object is as follows [13]:

- *Oj*: (feature value of the) ground object,
- *oid(Oj)*: object identifier,
- *d(Oj, P(Oj))*: distance of *Oj* to its parent.

The graphical representation of a ground object of Slim-Tree is given in Figure 4.4:

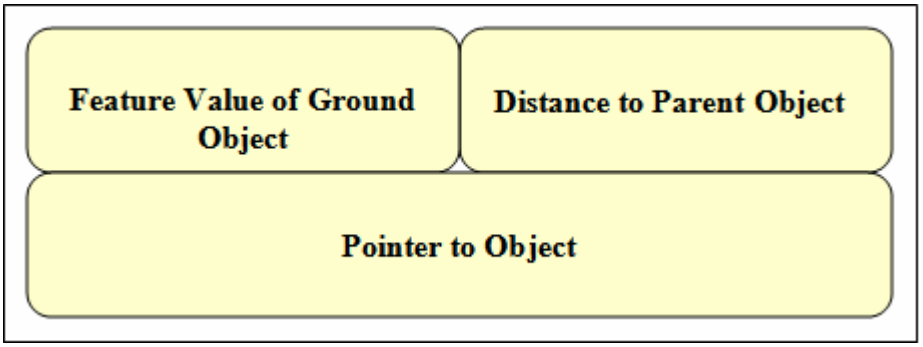

Figure 4.4 Structure of Ground Object in Slim-Tree

4.3 Building the Slim-Tree

When inserting a new object into Slim-Tree, at each level of the current tree, Slim-Tree tries to locate a node that covers the new object; if none qualifies, Slim-Tree selects the node whose center is nearest to the new object; if more than one node qualifies, and then Slim-Tree selects one of the candidate nodes by using

ChooseSubtree algorithm [3]. If the insertion of the new object causes an overflow on a node, then a new node is created at the same level and the objects on the overflowed node is splitted among these two nodes [3]. If the splitted node is the root node, then a new root node is allocated and the tree grows one level [3].

Slim-Tree has three options for the *ChooseSubtree* algorithm [3]:

- *Random*: randomly choose one of the qualifying nodes,
- *Mindist:* Choose the node that has the minimum distance from the new object and the center of the node,
- *Minoccup*: choose the node that has the minimum occupancy among the qualifying ones.

Slim-Tree has three splitting algorithms [3]:

- *Random*: The two new center objects are randomly selected, and the existing objects are distributed among them. Each object is stored in a new node that has its center nearest this object, with respect to a minimum utilization of each node.
- *minMax*: All possible pairs of objects are considered as potential representatives. For each pair, a linear algorithm assigns the objects to one of the representatives. The pair which minimizes the covering radius is chosen.
- *MST (Minimal Spanning Tree)*: The minimal spanning tree of the objects is generated, and one of the longest arcs of the tree is dropped. MST algorithm introduced in [3] is represented in Figure 4.5.

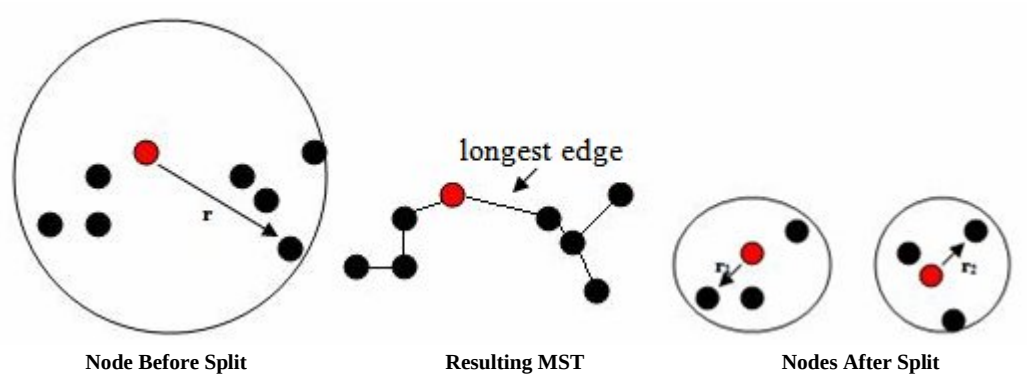

Figure 4.5 Sample Split using MST

As shown in Figure 4.5, during a split using MST in Slim-Tree, first the minimal spanning tree of the node is built. Then the longest edge of the minimal spanning tree is removed, the current connected objects are treated as two separate nodes and the representative objects (i.e. the object whose maximum distance to all other objects of the node is shortest) of the two nodes that are shown with red color are determined. This algorithm does not guarantee that each node will have a minimum percentage of objects [3]. Thus, as a solution to obtain more even distribution, choosing the most appropriate edge (i.e. the one which causes the best even distribution on other distributions) among the longest arcs is proposed in [3]. If no such edge exists for such even distribution, then the longest edge is removed as explained in Figure 4.5.

Since Slim-Tree is a metric tree, the relative distances between indexed objects are calculated to construct the Slim-Tree. In our system, the distance values between the low-level features (i.e. color layout, dominant color, region shape, edge histogram and motion activity) of any two video shots in Slim-Tree are computed using distance functions explained in Section 3.5.1, separately. These distance values are merged into one final distance value by using OWA operator as explained in Section 3.5.2.

Suppose that we construct a Slim-Tree using MST algorithm as split algorithm for seven video shots whose keyframes are as given in Figure 4.6.

Figure 4.6 Keyframes of Sample Seven Video Shots

During the insertion of an index entry for a video shot into the Slim-Tree, distance value between the video shot to be inserted and any video shot already in Slim-Tree is computed. Assume that, three video shots vs_1, vs_2 and vs_3 are already inserted into Slim-Tree, their relative distances are $d(v_1, v_2) = 0.149$, $d(v_1, v_3) = 0.324$, $d(v_2, v_3) = 0.335$, and the maximum capacity of a node is set to three. According to the given distance values, suppose that the data distribution for the current Slim-Tree is as in Figure 4.7 and the corresponding Slim-Tree representation is as in Figure 4.8.

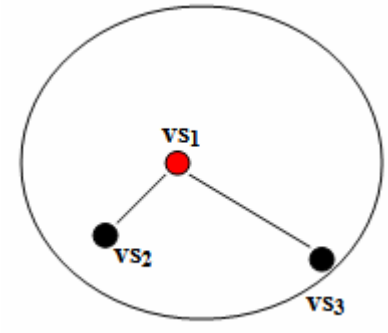

Figure 4.7 Sample Distribution of Three Video Shots

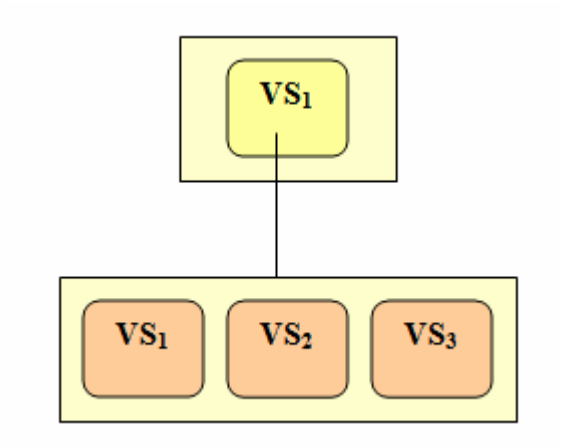

Figure 4.8 Corresponding Slim-Tree of Three Video Shots

We try to insert vs_4 to the Slim-Tree whose distribution is given in Figure 4.7. Since the maximum capacity of a node is three, the insertion of vs_4 causes a split. Distance values are computed as d (vs_4, vs_1) = 0.338, d (vs_4, vs_2) = 0.351 and d (vs_4, vs_3) = 0.259 using low-level features of video shots. Then the splitting steps of our current Slim-Tree are as follows:

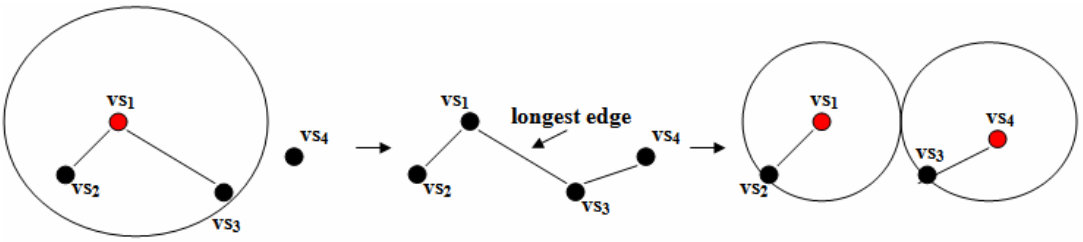

Figure 4.9 Splitting Slim-Tree of Video Shots using MST

As shown in Figure 4.9, firstly minimal spanning tree for four video shots is generated by computing distances between video shots. After minimal spanning tree is generated, it is detected that the edge between vs_1 and vs_3 in MST has the maximum distance value. Thus, vs_1-vs_3 edge is removed and currently connected video shots are assigned to the same node. After insertion of vs_4 and split operation using MST, the resulting Slim-Tree structure becomes as follows:

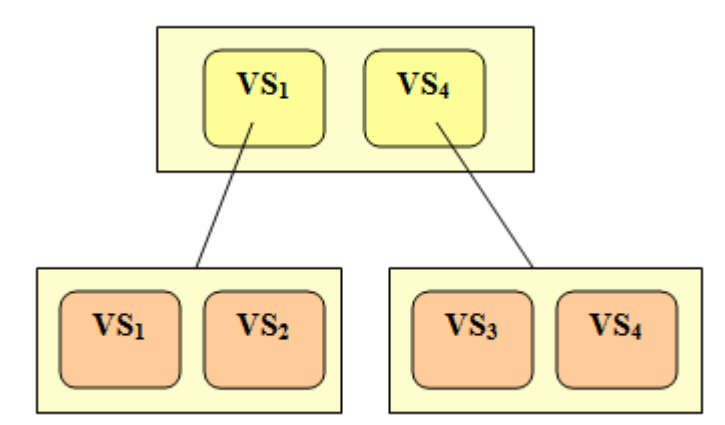

Figure 4.10 Resulting Slim-Tree of Four Video Shots after Split using MST

Slim-Tree introduces a new algorithm namely *Slim-down algorithm* [3] to reduce overlaps in an existing Slim-Tree in a post-processing step. The overlap in a metric space can be explained as the number of objects in the corresponding sub-trees which are covered by both nodes [3]. The Slim-down algorithm can be explained as follows [3] and it is represented in Figure 4.11:

1. For each node in a specified level of the tree, find the farthest object from the representative object.

2. Find a sibling node of the node, that also covers the farthest object. If such a node exists and has room for the farthest object, remove the farthest object from the node and insert into the sibling node and correct the radius of the node which the farthest node is removed.

3. Apply Step 1-2 sequentially, over all nodes of a specified level of the tree. If after a full round of the Step 1-2, an object moves from one node to another, another full round of Step 1-2 must be reapplied for the specified level.

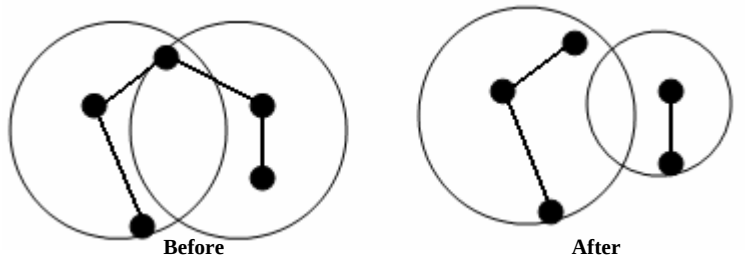

Figure 4.11 Slim-Down Algorithm

Suppose that we insert the remaining three video shots (vs_5, vs_6 and vs_7) to the Slim-Tree of four video shots mentioned above and our sample Slim-Tree that consists of seven video shots becomes as shown in Figure 4.12. In Section 4.4, we use this sample Slim-Tree of seven video shots to represent how k-NN and range queries are performed on Slim-Tree for video shots.

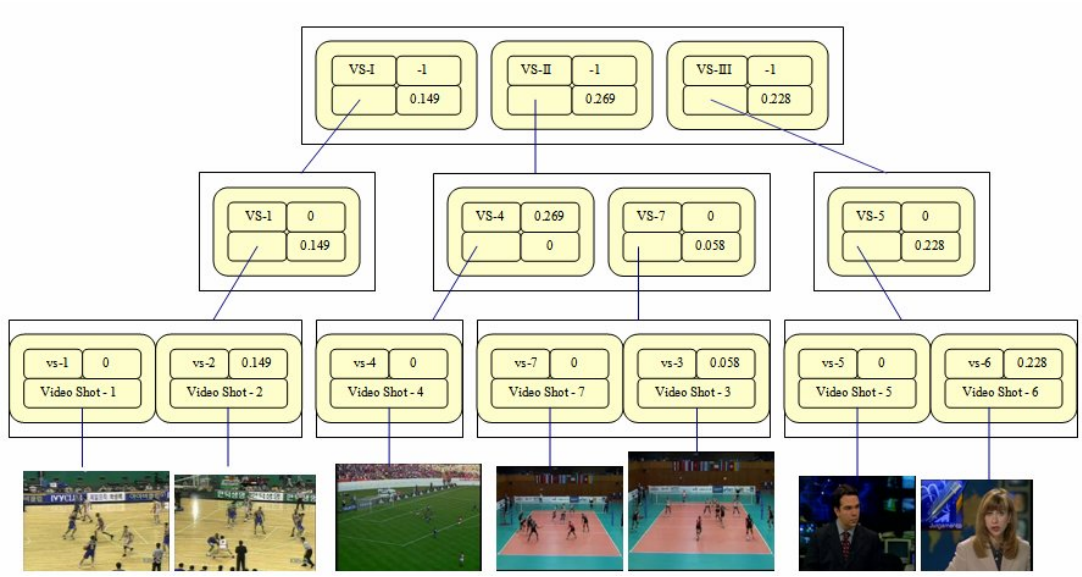

Figure 4.12 Sample Slim-Tree of Seven Video Shots

4.4 Querying the Slim-Tree

Slim-Tree supports two main types of queries: k-NN (k-Nearest Neighbor) query and range query. Since exact match query is a range query whose range radius is zero, it is not mentioned explicitly.

4.4.1 k-NN Query

Slim-Tree uses the triangle inequality to prune some nodes during k-NN queries, as other metric trees do. An example to the k-NN query of Slim-Tree is given below. Suppose that a query video shot, q whose keyframe is represented in Figure 4.13 is given to the system for retrieving the video shots which have the two closest distances to the query object (i.e. we make a 2-NN query) and our current Slim-Tree is the one given in Figure 4.12.

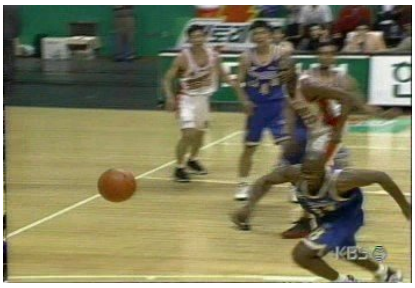

Figure 4.13 Keyframe of Given Query Video Shot for Retrieval

Firstly, the system computes the distances between the query video shot, *q*, and root node entries of the Slim-Tree. The distance values between the low-level features (i.e. color layout, dominant color, region shape, edge histogram and motion activity) of a video shot on Slim-Tree and q are computed using distance functions explained in Section 3.5.1, separately. These distance values are then merged into one distance value by using OWA operator explained in Section 3.5.2. The resulting distance value becomes the final distance value between the related video shot and q. Suppose that we compute the final distances as:

d(q, VS-I) = 0.192

d(q, VS-II) = 0.278

d(q, VS-III) = 0.353

Hence, d_k that is largest distance in current nearest neighbour objects becomes 0.278.

For the sub-tree of VS-I, the system decides whether to prune the sub-tree or not by using triangle inequality. For the first child of sub-tree of VS-I, which is VS-1 and equal to VS-I, triangle inequality is used to decide prune or not. If

$|d(q, VS\text{-}I) – d(VS\text{-}I, VS\text{-}1)| > d_k + r(VS\text{-}1)$

then the sub-tree of VS-1 can be pruned from results.

$d(VS\text{-}I, VS\text{-}1) = 0$

$|d(q, VS\text{-}I) – d(VS\text{-}I, VS\text{-}1)| > d_k + r(VS\text{-}1)$

$|0.192 – 0| > 0.278 + 0.149$

$0.192 > 0.427$ (not true)

So, d(q, vs-1) and d(q, vs-2) have to be calculated, which is equal to 0.192 and 0.205, respectively. d(q, vs-1) becomes our *best-so-far* and d(q, vs-2) becomes our *second-best-so-far*. Since d(q, vs-2) is less than current d_k, d_k is updated as 0.205.

For the sub-tree of VS-II, the system decides whether to prune the sub-tree or not by using triangle inequality. For the first child of VS-II, which is VS-4, triangle inequality is used to decide prune or not. If

$|d(q, VS\text{-}II) – d(VS\text{-}II, VS\text{-}4)| > d_k + r(VS\text{-}4)$

then the sub-tree of VS-4 can be pruned from results.

$d(VS\text{-}II, VS\text{-}4) = 0.269$

$|d(q, VS\text{-}II) – d(VS\text{-}II, VS\text{-}4)| > d_k + r(VS\text{-}4)$

$|0.278 – 0.269| > 0.205 + 0.0$

$0.009 > 0.205$ (not true)

So, d(q, vs-4) has to be calculated, which is equal to 0.296. Since d(q, vs-4) is greater than d_k, then vs-4 is eliminated from the results.

For the second entry of sub-tree of VS-II, which is VS-7 and equal to VS-II, same procedure is applied to decide pruning.

$d(VS\text{-}II, VS\text{-}7) = 0.0$

$|d(q, VS\text{-}II) – d(VS\text{-}II, VS\text{-}7)| > d_k + r(VS\text{-}7)$

$|0.278 – 0.0| > 0.205 + 0.058$

$0.278 > 0.263$ (true)

So, the sub-tree of VS-7 can be pruned from results.

Finally, for the sub-tree of VS-III, the system decides whether to prune the sub-tree or not by using triangle inequality. For the child of VS-III, which is VS-5 and equal to VS-III, triangle inequality is used to decide prune or not. If

$|d(q, VS\text{-}III) – d(VS\text{-}III, VS\text{-}5)| > d_k + r(VS\text{-}5)$

then the sub-tree of VS-5 can be pruned from results.

$d(VS\text{-}III, VS\text{-}5) = 0$

$|d(q, VS\text{-}III) – d(VS\text{-}III, VS\text{-}5)| > d_k + r(VS\text{-}5)$

$|0.353 – 0| > 0.205 + 0.228$

$0.353 > 0.433$ (not true)

So, d(q, vs-5) and d(q, vs-6) have to be calculated, which is equal to 0.353 and 0.575, respectively. Since d(q, vs-5) and d(q, vs-6) are greater than d_k, then vs-5 and vs-6 are eliminated from the results.

As a result, vs-1 and vs-2 are returned to the user, since d(q, vs-1) and d(q, vs-2) have the two smallest distances to the query video shot.

4.4.2 Range Query

Slim-Tree uses the triangle inequality to prune some nodes during range queries, as other metric trees do. An example to the range query of Slim-Tree is given below. Suppose that a query video shot, q whose keyframe is represented in Figure 4.13 is given to the system for retrieving the video shots which have a distance from the query video shot less than or equal 0.21 (r = 0.21) and our current Slim-Tree is the one given in Figure 4.12.

Firstly, as in k-NN query of Slim-Tree, the system computes the distances between the query video shot, *q*, and root node entries of the Slim-Tree. The distance value between any node entry of Slim-Tree and q is computed as follows. The distance values between the low-level features (i.e. color layout, dominant color, region shape, edge histogram and motion activity) of a video shot on Slim-Tree and q are computed using distance functions explained in Section 3.5.1, separately. These distance values are then merged into one distance value by using OWA operator explained in Section 3.5.2. The resulting distance value becomes the final distance value between the related video shot and q. Suppose that the final distances are:

d(q, VS-I) = 0.192

d(q, VS-II) = 0.278

d(q, VS-III) = 0.353

For the sub-tree of VS-I, the system decides whether to prune the sub-tree or not by using triangle inequality. For the first child of VS-I, which is VS-1 and equal to VS-I, triangle inequality is used to decide prune or not. If

$|d(q, VS\text{-}I) - d(VS\text{-}I, VS\text{-}1)| > r(q) + r(VS\text{-}1)$

then the sub-tree of VS-1 can be pruned from results.

d(VS-I, VS-1) = 0

$|d(q, VS\text{-}I) - d(VS\text{-}I, VS\text{-}1)| > r(q) + r(VS\text{-}1)$

$|0.192 – 0| > 0.21 + 0.149$

$0.192 > 0.359$ (not true)

So, d(q, vs-1) and d(q, vs-2) have to be calculated, which is equal to 0.192 and 0.205, respectively. Since d(q, vs-1) and d(q, vs-2) are less than 0.21, then vs-1 and vs-2 are added to the results.

For the sub-tree of VS-II, the system decides whether to prune the sub-tree or not by using triangle inequality. For the first child of VS-II, which is VS-4, triangle inequality is used to decide prune or not. If

$|d(q, VS\text{-}II) – d(VS\text{-}II, VS\text{-}4)| > r(q) + r(VS\text{-}4)$

then the sub-tree of VS-4 can be pruned from results.

$d(VS\text{-}II, VS\text{-}4) = 0.269$

$|d(q, VS\text{-}II) – d(VS\text{-}II, VS\text{-}4)| > r(q) + r(VS\text{-}4)$

$|0.278 – 0.269| > 0.21 + 0.0$

$0.009 > 0.21$ (not true)

So, d(q, vs-4) has to be calculated, which is equal to 0.296. Since d(q, vs-4) is greater than 0.21, then vs-4 is eliminated from the results.

For the second entry of sub-tree of VS-II, which is VS-7 and equal to VS-II, same procedure is applied to decide pruning.

$d(VS\text{-}II, VS\text{-}7) = 0.0$

$|d(q, VS\text{-}II) – d(VS\text{-}II, VS\text{-}7)| > r(q) + r(VS\text{-}7)$

$|0.278 – 0.0| > 0.21 + 0.058$

$0.278 > 0.268$ (true)

So, the sub-tree of VS-7 can be pruned from results without further processing (i.e. distance computation)

Finally, for the sub-tree of VS-III, the system decides whether to prune the sub-tree or not by using triangle inequality. For the child of VS-III, which is VS-5 and equal to VS-III, triangle inequality is used to decide prune or not. If

$|d(q, VS\text{-}III) – d(VS\text{-}III, VS\text{-}5)| > r(q) + r(VS\text{-}5)$

then the sub-tree of VS-5 can be pruned from results.

$d(VS\text{-}III, VS\text{-}5) = 0$

$|d(q, VS\text{-}III) – d(VS\text{-}III, VS\text{-}5)| > r(q) + r(VS\text{-}5)$

$|0.353 – 0| > 0.21 + 0.228$

$0.353 > 0.438$ (not true)

So, d(q, vs-5) and d(q, vs-6) have to be calculated, which is equal to 0.353 and 0.575, respectively. Since d(q, vs-5) and d(q, vs-6) are greater than 0.21, then vs-5 and vs-6 are eliminated from the results.

As a result, vs-1 and vs-2 are returned to the user, since d(q, vs-1) and d(q, vs-2) are less than 0.21.

CHAPTER 5

BITMATRIX

In this chapter, the index structure adapted to our content-based retrieval system, namely BitMatrix is discussed, in detail.

5.1 Introduction

BitMatrix [1][2] is an index structure proposed for similarity searching of multimedia data by using a data approximation approach. To index the low-level features of multimedia data, BitMatrix uses a data approximation approach in the spirit of VA-File described in Section 2.2.4. BitMatrix partitions each dimension *D* of feature vectors of multimedia data into a set of ranges $\pi_D = \{r_i = [l_i, u_i], i = 1 \ldots k_D\}$, where l_i and u_i are the lower and upper bound of range r_i and assigns the multimedia object to the range where it belongs in the related feature dimension.

Various partitioning schemes can be used to partition feature vectors of multimedia data such as *equi-width, equi-depth* or *k-means* clustering. In *equi-width* clustering, the width of each range in a dimension is equal to each other, where in *equi-depth* clustering, the number of indexed objects in each range of a dimension is equal to each other. Finally, with *k-means* clustering, a dimension is divided into a pre-defined number of ranges and the indexed objects are assigned to an appropriate range. The generated approximations of feature vectors of multimedia data are used during queries to prune the search space effectively with a sequential analysis.

5.2 Building the BitMatrix

Before the discussion on building BitMatrix, we give some definitions that exist in BitMatrix terminology.

Bitmap Signature [1] – Given a partitioning scheme such as equi-width, equi-depth or k-means clustering, an object's bitmap signature is a bit string of length $\sum_{D=1}^{N} k_D$, where N is the number of dimensions and k_D is the number of ranges for a specific dimension D. For each dimension D, the signature contains 1 for the range where the object belongs and 0 for the other ranges. Suppose that there are two feature dimensions that are clustered. First dimension is clustered into three and second dimension is clustered into four dimensions, then the bitmap signature of an object, o_1 becomes 0010001 if the object is in the thirth cluster of first dimension and in the fourth cluster of the second dimension.

Cardinality of Bitmap Signature [1] – The number of bits set to 1 in the bitmap signature of an object. The cardinality of the bitmap signature of the object o_1 given as an example above is 2.

Bitmatrix [1] – The index structure that contains bitmap signatures of each indexed object that are arranged as lines in a matrix structure. An example BitMatrix of six objects is given in Table 5.1.

Algorithm-1: Build BitMatrix Algorithm - Generic

INPUT: Number of Feature Dimensions – N
Number of Objects - M
Feature Vectors of Objects - FeatureVectors[N][M]
OUTPUT: Bitmap signatures of the Objects, BitMatrix.

```
Begin
  Initialize BM to empty BitMatrix with M rows and ∑_{D=1}^{N} k_D columns
  for int i = 1 to N
    Apply clustering algorithm on FeatureVectors[i] to divide FeatureVectors[i] into k_i ranges
    for  int j = 1 to M
      Assign 1 for the range where FeatureVectors[i][j] belongs in BM
      Assign 0 for the other ranges in BM
    end for
  end for
  return BM
End
```

The first step to build a BitMatrix is clustering feature vectors of multimedia objects to be indexed along each dimension D into k_d ranges. Each feature dimension is clustered seperately by using a clustering algorithm such as k-means. After each feature dimension of multimedia objects is clustered, bitmap signature of each multimedia object is computed by using the clustering results and put into the BitMatrix.

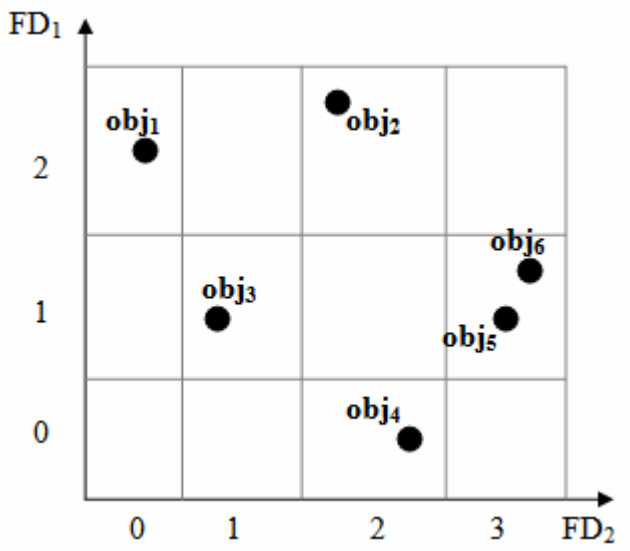

Figure 5.1 Sample Clustering of Six Objects along FD_1 and FD_2 Dimensions

Suppose that, we have six multimedia objects to be indexed along their two dimensions, namely FD_1 and FD_2. We can think these two dimensions for example, color and texture feature dimensions of images or color and motion feature dimensions of videos etc. Figure 5.1 represents the clustering results of the feature vectors of these six multimedia objects. As seen from Figure 5.1, objects are clustered into three ranges along FD_1 and into four ranges along FD_2. According to these clustering results, the resulting BitMatrix index structure is as follows:

Table 5.1 Resulting BitMatrix for Six Objects

	FD_1			**FD_2**			
	0	***1***	***2***	***0***	***1***	***2***	***3***
obj1	0	0	**1**	**1**	0	0	0
obj2	0	0	**1**	0	0	**1**	0
obj3	0	**1**	0	0	**1**	0	0
obj4	**1**	0	0	0	0	**1**	0
obj5	0	**1**	0	0	0	0	**1**
obj6	0	**1**	0	0	0	0	**1**

The algorithm for the construction of BitMatrix for video shots in our CBR system is given below.

Algorithm-2: Build BitMatrix for Video Shots Algorithm

INPUT: XML database environment to connect – Path2DBEnv
OUTPUT: Bitmap signatures of the video shots in specified XML database, BitMatrix.

```
Begin
  Initialize BM to empty BitMatrix

  Fetch ColorLayout, DominantColor, RegionShape, EdgeHistogram and MotionActivity
  descriptors of Video Shots from XML DB

  Assign number of video shots to M

  Cluster ColorLayout descriptors using k-means algorithm
  for  int j = 1 to M
       Assign 1 for the range where ColorLayout[j] belongs in BM
       Assign 0 for the other ranges in BM
  end for

  Cluster DominantColor descriptors using k-means algorithm
  for  int j = 1 to M
       Assign 1 for the range where DominantColor[j] belongs in BM
       Assign 0 for the other ranges in BM
  end for

  Cluster RegionShape descriptors using k-means algorithm
   for  int j = 1 to M
       Assign 1 for the range where RegionShape[j] belongs in BM
       Assign 0 for the other ranges in BM
  end for

  Cluster EdgeHistogram descriptors using k-means algorithm
  for  int j = 1 to M
       Assign 1 for the range where EdgeHistogram[j] belongs in BM
       Assign 0 for the other ranges in BM
  end for

  Cluster MotionActivity descriptors using k-means algorithm
   for  int j = 1 to M
       Assign 1 for the range where MotionActivity[j] belongs in BM
       Assign 0 for the other ranges in BM
  end for

  return BM
End
```

As explained in Algorithm-2, using k-means clustering algorithm the system clusters low-level visual features (i.e. color layout, dominant color, edge histogram, region shape and motion activity) of video shots seperately into a pre-defined number of clusters to construct

BitMatrix for video shots. BitMatrix index structure is then generated by using clustering results of each low-level feature.

To clarify the adaptation of BitMatrix for indexing low-level feature vectors of video shots, a sample BitMatrix built operation for six video shots is explained in the following section. The keyframes for the six video shots are as follows:

Figure 5.2 Keyframes of Sample Six Video Shots

Clustering results of six video shots along five dimensions are given in Figure 5.3, Figure 5.4 and Figure 5.5. The resulting BitMatrix index structure is given in Table 5.2. The five dimensions used for clustering video shots are Color Layout (CL), Dominant Color (DC), Region Shape (RS), Edge Histogram (EH) and Motion Activity (MA) low-level descriptors of video shots.

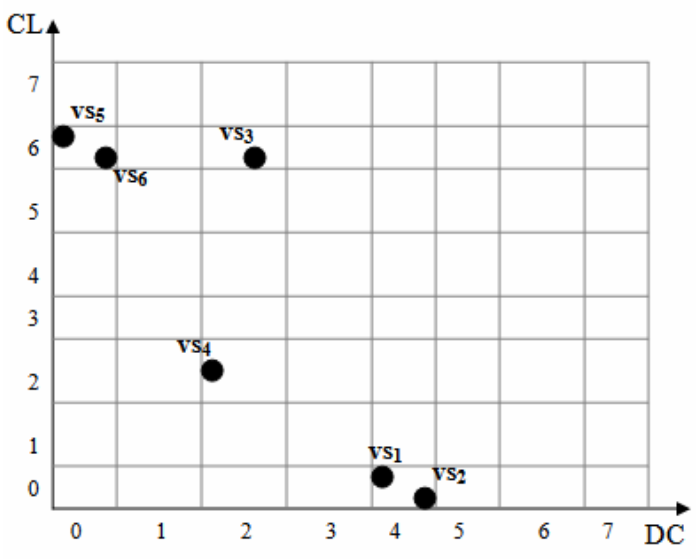

Figure 5.3 Sample Clustering of Six Video Shots along CL and DC Dimensions

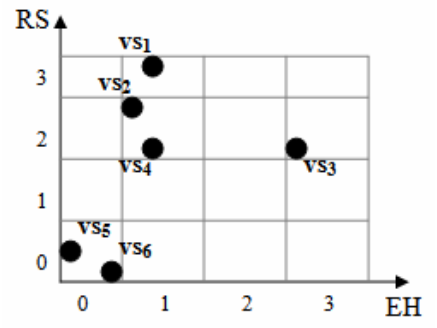

Figure 5.4 Sample Clustering of Same Six Video Shots along RS and EH Dimensions

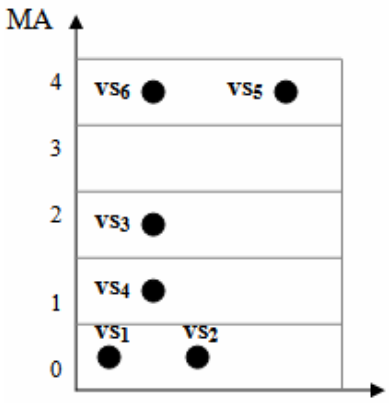

Figure 5.5 Sample Clustering of Same Six Video Shots along MA Dimension

As seen from Figure 5.3, Figure 5.4 and Figure 5.5, six video shots are clustered into eight ranges along CL and DC, into four ranges along RS and EH and into five ranges along MA dimension.

Table 5.2 Resulting BitMatrix for the Six Video Shots

	CL								*DC*								*RS*				*EH*				*MA*				
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*0*	*1*	*2*	*3*	*0*	*1*	*2*	*3*	*0*	*1*	*2*	*3*	*4*
vs_1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0
vs_2	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0
vs_3	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0
vs_4	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0
vs_5	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1
vs_6	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1

As seen from the sample BitMatrix (Table 5.2), bitmap signature of an indexed object contains "1" for the range where the object belongs and "0" for the other ranges. In addition, it is seen from Table 5.2 that the size of BitMatrix index grows linearly with the number of indexed objects and with the dimensionality.

Algorithm-3: Insert into BitMatrix Algorithm

INPUT: Number of Feature Dimensions – N
Feature Vectors of Object to be inserted – FeatureVectors[N]
Current BitMatrix – BM
OUTPUT: Updated BitMatrix.

```
Begin
  for int i = 1 to N
    Apply clustering algorithm on FeatureVectors[i] to find the range that FeatureVectors[i]
    belongs
    Assign 1 for the range where FeatureVectors[i] belongs in BM
    Assign 0 for the other ranges in BM
  end for
  return BM
End
```

To insert an object into BitMatrix, firstly the bitmap signature of the object to be indexed is computed by using clustering. The resulting bitmap signature of the object is added to the end of BitMatrix as a new line.

Algorithm-4: Delete from BitMatrix Algorithm

INPUT: Current BitMatrix – BM
Index of Object to be deleted in BM – index
OUTPUT: Success/Failure.

```
Begin
  bool result = Remove bitmap signature from BM using index
  return result
End
```

To delete an object from BitMatrix, firstly the bitmap signature of the object to be deleted is located on BitMatrix. The row that contains the bitmap signature is deleted from the BitMatrix. To locate the row index of the bitmap signature of an object, a mapping between the indexed objects and related bitmap signatures is needed. In our system, we provide this mapping by using the name of the indexed video shots, since the name of the video shots implicitly includes the row index of the related bitmap signature (e.g. 250.mpg is a video shot at 250^{th} row on BitMatrix).

Algorithm-5: Update Bitmap Signature Algorithm

INPUT: Number of Feature Dimensions – N
Feature Vectors of Object to be updated – FeatureVectors[N]
Current BitMatrix – BM
Index of Object in BM – index
OUTPUT: Updated BitMatrix.

```
Begin
  for int i = 1 to N
    Apply clustering algorithm on FeatureVectors[i] to find the range that FeatureVectors[i]
    belongs
    Assign 1 for the range where FeatureVectors[i] belongs in BM
    Assign 0 for the other ranges in BM
  end for
  return BM
End
```

To update an object on BitMatrix, firstly the bitmap signature of the object to be updated is located on BitMatrix. The row that contains the bitmap signature is updated with the new bitmap signature of the object on the BitMatrix using clustering. In our system, the mapping between the indexed objects and related bitmap signatures are provided by using the name of the indexed video shots as in deletion case.

5.3 Querying the BitMatrix

BitMatrix supports two main types of queries: k-NN (k-Nearest Neighbor) query and range query. Since exact match query is a range query whose range radius is zero, it is not mentioned explicitly.

5.3.1 k-NN Query

5.3.1.1 Naïve Approach

Algorithm-6: k-NN Query on BitMatrix (Naïve Approach)

```
INPUT: Number of Feature Dimensions – N
       Number of Objects – M
       Feature Vectors of Indexed Objects – ObjectFeatureVectors[M][N]
       Feature Vectors of Query Object – QFeatureVector[N]
       Current BitMatrix – BM
       Cardinality Threshold – ct
       Number of Nearest Neighbour to Return - k
OUTPUT: k nearest neighbour objects of the query object.

Begin
  Initialize BS to empty bitmap signature
  Initialize DistanceFeatureList to empty list
  Initialize RemainingObjects to empty list

  for int i = 1 to N
    Apply clustering algorithm on QFeatureVector[i] to find the range that QFeatureVector[i]
    belongs
    Assign 1 for the range where QFeatureVector[i] belongs in BS
    Assign 0 for the other ranges in BS
  end for

  for int i = 1 to BM.length
    ResultingBitmapSignature = Perform Bitwise AND between BS and BM[i]
    cardinality = Compute the cardinality of ResultingBitmapSignature
    if cardinality > ct then
      Add ObjectFeatureVectors[i] to RemainingObjects
    endif
  end for

  for int i = 1 to RemainingObjects.length
    distance = Compute exact distance between ObjectFeatureVectors[i] and QFeatureVector[N]
    Add distance and ObjectFeatureVectors[i] to DistanceFeatureList
  end for

  Sort DistanceFeatureList in Ascending Order according to distance values
  return DistanceFeatureList.subList(1, k)
End
```

k-NN query on BitMatrix using naïve approach [1] can be explained as follows. Given a query object, *q* with feature vectors QFeatureVector[N] where N is the number of feature dimensions. Firstly, we obtain the bitmap signature of q using QFeatureVector[N] (i.e. for each feature dimension D, find the range in which the QFeatureVector[D] lies). Iterate through the bitmap signatures on BitMatrix performing bitwise AND with the query object's bitmap signature. If the cardinality of a resulting bitwise ANDed bitmap signature is above the pre-defined *cardinality threshold (ct)*, the related object is retained for the next phase. In the next phase, feature vectors of the remaining objects are accessed. For each remaining object, the exact distance to the query object is computed using feature vectors. Finally, *k* objects that are closest to the query object are returned as the results.

5.3.1.1.1 Retrieval of Video Shots using Naïve Approach

In this section, it is explained how k-NN query on BitMatrix using naïve approach is performed for video shots. Suppose that, we are given a video shot (q_{naive}) whose keyframe is as follows:

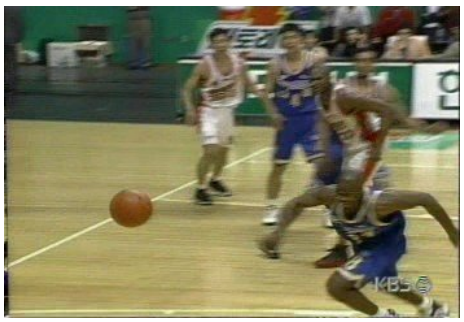

Figure 5.6 Keyframe of Given Query Video Shot (q_{naive}) for k-NN Query using BitMatrix

Also, suppose that our current BitMatrix is the BitMatrix for the six video shots given in Table 5.2 and we make 2-NN query. Firstly, we compute the bitmap signature of the given query video shot (q_{naive}) and find the bitmap signature of q_{naive} as 10000000000000100001010001000. The elimination phase of the k-NN query on BitMatrix using naïve approach is as follows.

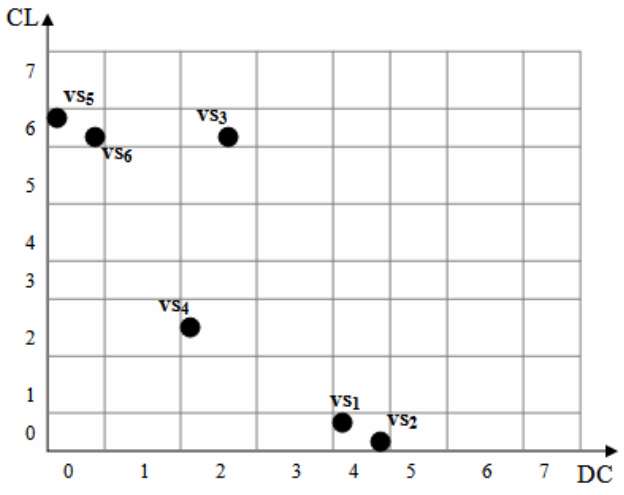

Figure 5.7 Clustering Results of Six Video Shots along CL and DC Dimensions

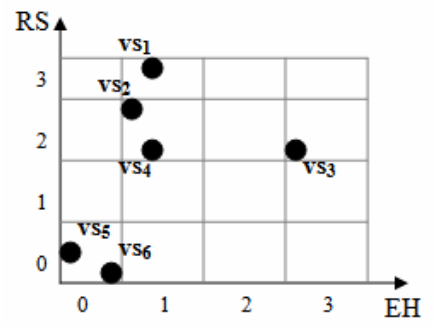

Figure 5.8 Clustering Results of Six Video Shots along RS and EH Dimensions

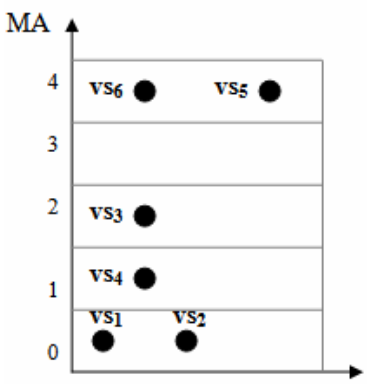

Figure 5.9 Clustering Results of Six Video Shots along MA Dimension

Table 5.3 Bitwise AND with Query Video Shot

		vs_1	vs_2	vs_3	vs_4	vs_5	vs_6	q_{naive}
CL	***0***	**1**	**1**	0	0	0	0	**1**
	1	0	0	0	0	0	0	0
	2	0	0	0	**1**	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
	6	0	0	**1**	0	**1**	**1**	0
	7	0	0	0	0	0	0	0
DC	***0***	0	0	0	0	**1**	**1**	0
	1	0	0	0	0	0	0	0
	2	0	0	**1**	**1**	0	0	0
	3	0	0	0	0	0	0	0
	4	**1**	**1**	0	0	0	0	0
	5	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	**1**
	7	0	0	0	0	0	0	0
RS	***0***	0	0	0	0	**1**	**1**	0
	1	0	0	0	0	0	0	0
	2	0	**1**	**1**	**1**	0	0	0
	3	**1**	0	0	0	0	0	**1**
EH	***0***	0	0	0	0	**1**	**1**	0
	1	**1**	**1**	0	**1**	0	0	**1**
	2	0	0	0	0	0	0	0
	3	0	0	**1**	0	0	0	0
MA	***0***	**1**	**1**	0	0	0	0	0
	1	0	0	0	**1**	0	0	**1**
	2	0	0	**1**	0	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	0	0	**1**	**1**	0

Table 5.4 Resulting Bitmap Signatures and Cardinalities After Bitwise AND Operations

		q_{naive} & vs_1	q_{naive} & vs_2	q_{naive} & vs_3	q_{naive} & vs_4	q_{naive} & vs_5	q_{naive} & vs_6
CL	***0***	**1**	**1**	0	0	0	0
	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0
	5	0	0	0	0	0	0
	6	0	0	0	0	0	0
	7	0	0	0	0	0	0
DC	***0***	0	0	0	0	0	0
	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0
	5	0	0	0	0	0	0
	6	0	0	0	0	0	0

		q_{naive} & vs_1	q_{naive} & vs_2	q_{naive} & vs_3	q_{naive} & vs_4	q_{naive} & vs_5	q_{naive} & vs_6
	7	0	0	0	0	0	0
RS	***0***	0	0	0	0	0	0
	1	0	0	0	0	0	0
	2	0	0	0	0	0	0
	3	**1**	0	0	0	0	0
EH	***0***	0	0	0	0	0	0
	1	**1**	**1**	0	**1**	0	0
	2	0	0	0	0	0	0
	3	0	0	0	0	0	0
MA	***0***	0	0	0	0	0	0
	1	0	0	0	**1**	0	0
	2	0	0	0	0	0	0
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0
Cardinality		**3**	**2**	**0**	**2**	**0**	**0**

After the bitmap signature of q_{naive} is computed, the bitmap signature of q_{naive} is bitwise ANDed with the bitmap signatures of video shots on BitMatrix as shown in Table 5.3 and Table 5.4. Table 5.4 shows the cardinality results after bitwise AND operations. Suppose that the cardinality threshold is set to two, then vs_1, vs_2 and vs_4 are retained to the last phase (i.e. exact distance computation phase) of the k-NN search algorithm.

In the exact distance computation phase, we compute the distances between the low-level features (i.e. color layout, dominant color, region shape, edge histogram and motion activity) of vs_1, vs_2 and vs_4 and the q_{naive}'s corresponding features using distance functions explained in Section 3.5.1. For each low-level feature, distance value between a video shot (vs_1, vs_2 or vs_4) and q_{naive} is computed, seperately. These distance values are merged into one distance value by using OWA operator explained in Section 3.5.2. The resulting distance value becomes the final distance value between the video shot (vs_1, vs_2 or vs_4) and q_{naive}. Finally, we find the exact distances as follows:

$d(q_{naive}, vs_1) = 0.192$

$d(q_{naive}, vs_2) = 0.205$

$d(q_{naive}, vs_4) = 0.296$

Since we make 2-NN query, then vs_1 and vs_2 are returned as results.

5.3.1.2 Using Range Expansion Heuristic

The naïve approach sometimes may eliminate objects that happens to be one of the nearest neighbors in a k-NN search [1]. This effect, known as the *edge-effect* appears because for all dimensions D, only the objects in the same range as query object's are considered [1]. Thus, in order to reduce the edge-effect, *range-expansion* [1] heuristic is applied to the naïve approach. The heuristic modifies the elimination phase of the naïve approach as follows:

Algorithm-7: k-NN Query on BitMatrix (Range Expansion Applied)

INPUT: Number of Feature Dimensions – N
Number of Objects – M
Feature Vectors of Indexed Objects – ObjectFeatureVectors[M][N]
Feature Vectors of Query Object – QFeatureVector[N]
Current BitMatrix – BM
Cardinality Threshold – ct
Number of Nearest Neighbour to Return - k
OUTPUT: k nearest neighbour objects of the query object.

```
Begin
   … // Same as in naïve approach

  for int i = 1 to N
    Apply clustering algorithm on QFeatureVector[i] to find the range, r_i, that QFeatureVector[i]
    belongs

    Assign 1 for the range where QFeatureVector[i] belongs in BS
    Assign 0 for the other ranges in BS

    //Apply range expansion heuristic
    l_ri = lower bound of r_i
    u_ri = upper bound of r_i
    et_i = expansion threshold for i^th dimension

    if QFeatureVector[i] - l_ri < et_i * (u_ri - l_ri) then
       Assign 1 for the range r_i-1 in BS
    endif

    if QFeatureVector[i] - u_ri < et_i * (u_ri - l_ri) then
       Assign 1 for the range r_i+1 in BS
    endif
  end for
  … // Same as in naïve approach
End
```

Assume that a query object, q, lies in range r_i for dimension i and q_i represents the feature vector of the query object for dimension i. The expansion takes place to the left if $q_i - l_{ri} < et_i$

* (u_{ri} - l_{ri}) or to the right if q_i - u_{ri} < et_i * (u_{ri} - l_{ri}), where u_{ri} and l_{ri} are the upper and lower bounds of the r_i^{th} range for dimension *i* respectively and et_i is the *expansion threshold* [1] value for dimension *i* that can be in range [0, 0.5].

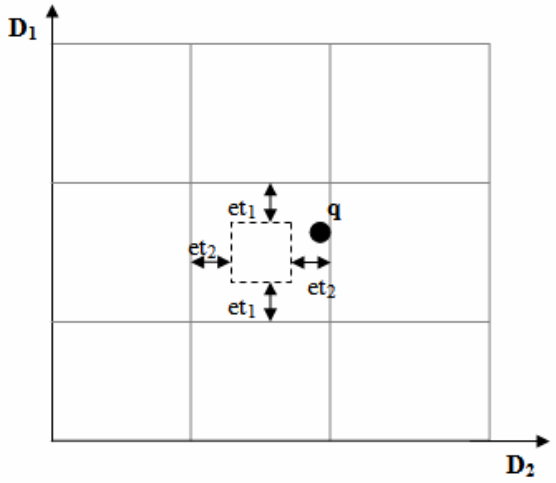

Figure 5.10 Range-Expansion Heuristic

In Figure 5.10, et_1 is the expansion threshold of D_1 and et_2 is the expansion threshold of D_2. If the query object, *q* is closer to the right/left edge of the range of a dimension than the pre-defined expansion threshold (et_1/et_2) of the related dimension (D_1/D_2), then it's bitmap signature is expanded to the right/left (i.e. the range right/left next to it also set to 1), respectively.

5.3.1.2.1 Retrieval of Video Shots using Range Expansion

Range expansion heuristic is applied in our content-based retrieval system for videos. k-NN search using range expansion is explained in this section to show the effects of using this heuristic for videos. Suppose that, we are given the video shot, q_{exp} whose keyframe is given in Figure 5.6 and our current BitMatrix is the BitMatrix for the six video shots given in Table 5.2. Also, suppose that we make 2-NN query again and q_{exp} is close to the left edge of its range than et_{RS} in Region Shape (RS) dimension, where et_{RS} is the expansion threshold for dimension RS.

Firstly, we compute the bitmap signature of the q_{exp} and find the bitmap signature of q_{exp} as 1000000000000010001010001000. The elimination phase of the k-NN query on BitMatrix using range expansion is as follows:

Table 5.5 Cardinality Results After Bitwise AND Operations (Range-Expansion Applied)

		vs_1	vs_2	vs_3	vs_4	vs_5	vs_6	q_{exp}
CL	***0***	**1**	**1**	0	0	0	0	**1**
	1	0	0	0	0	0	0	0
	2	0	0	0	**1**	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	0	0	0	0	0
	5	0	0	0	0	0	0	0
	6	0	0	**1**	0	**1**	**1**	0
	7	0	0	0	0	0	0	0
DC	***0***	0	0	0	0	**1**	**1**	0
	1	0	0	0	0	0	0	0
	2	0	0	**1**	**1**	0	0	0
	3	0	0	0	0	0	0	0
	4	**1**	**1**	0	0	0	0	0
	5	0	0	0	0	0	0	0
	6	0	0	0	0	0	0	**1**
	7	0	0	0	0	0	0	0
RS	***0***	0	0	0	0	**1**	**1**	0
	1	0	0	0	0	0	0	0
	2	0	**1**	**1**	**1**	0	0	**1**
	3	**1**	0	0	0	0	0	**1**
EH	***0***	0	0	0	0	**1**	**1**	0
	1	**1**	**1**	0	**1**	0	0	**1**
	2	0	0	0	0	0	0	0
	3	0	0	**1**	0	0	0	0
MA	***0***	**1**	**1**	0	0	0	0	0
	1	0	0	0	**1**	0	0	**1**
	2	0	0	**1**	0	0	0	0
	3	0	0	0	0	0	0	0
	4	0	0	0	0	**1**	**1**	0
Cardinality		**3**	**3**	**1**	**3**	**0**	**0**	

Suppose that the cardinality threshold is set to three and we use naive approach, then only vs_1 is retained to the last phase (i.e. exact distance computation phase) of the k-NN search algorithm and vs_2 that is also similar to q_{exp} is eliminated. But, if we use range expansion heuristic, then the cardinality of vs_2 also becomes 3 as shown in Table 5.5 and is retained to the exact distance computation phase of the k-NN search. The exact distance computation phase of the k-NN search using range expansion heuristic is carried out as explained in Section 5.3.1.1.1.

5.3.2 Range Query

Since range query is very similar to k-NN query for BitMatrix and k-NN query algorithms are discussed in Section 5.3.1 in detail, we do not give further explanation on range query of BitMatrix other than its basic algorithm. The only part that changes in a range query is colored with red in Algorithm-8. Algorithm-8 gives range query using naive approach.

However, range expansion heuristic can also be applied on range query, since the elimination step of both k-NN query and range query is same.

Algorithm-8: Range Query on BitMatrix (Naïve Approach)

```
INPUT: Number of Feature Dimensions – N
       Number of Objects – M
       Feature Vectors of Indexed Objects – ObjectFeatureVectors[M][N]
       Feature Vectors of Query Object – QFeatureVector[N]
       Current BitMatrix – BM
       Cardinality Threshold – ct
       Range radius - r
OUTPUT: Neighbour objects whose distances are smaller than or equal to r to the query object.

Begin
  Initialize BS to empty bitmap signature
  Initialize RemainingObjects to empty list
  Initialize ResultSet to empty list

  for int i = 1 to N
    Apply clustering algorithm on QFeatureVector[i] to find the range that QFeatureVector[i]
    belongs
    Assign 1 for the range where QFeatureVector[i] belongs in BS
    Assign 0 for the other ranges in BS
  end for

  for int i = 1 to BM.length
    ResultingBitmapSignature = Perform Bitwise AND between BS and BM[i]
    cardinality = Compute the cardinality of ResultingBitmapSignature
    if cardinality > ct then
      Add ObjectFeatureVectors[i] to RemainingObjects
    endif
  end for

  for int i = 1 to RemainingObjects.length
    distance = Compute exact distance between ObjectFeatureVectors[i] and QFeatureVector[N]
    if distance <= r then
      Add RemainingObjects[i] to ResultSet
    endif
  end for

  return ResultSet
End
```

As a final word on BitMatrix, we can say that BitMatrix is a highly parametrizable index structure and therefore there is a tradeoff between precision and speed that is controlled by various parameters such as cardinality threshold and expansion threshold.

CHAPTER 6

IMPLEMENTATION

In this chapter, the implementation details of our content-based retrieval system introduced in Chapter 3 and tools/APIs used during the implementation are discussed, in detail.

6.1 Video Shot Detection and Keyframe Extraction

In this study, we use IBM VideoAnnEx Annotation Tool [32] for the detection of the video shots in given sequential video streams and for the extraction of the keyframes for each video shot. The tool segments a video sequence into its shots by detecting scene cuts, dissolutions and fading. Also, the tool is able to extract keyframes for the detected video shots. In this study, we use one keyframe to represent the still image low-level features (i.e. color layout, dominant color, region shape and edge histogram) of each detected video shot, since IBM VideoAnnEx Annotation Tool provides one keyframe for each video shot.

6.2 Feature Extraction

Low-level features of video shots (i.e. color layout, dominant color, edge histogram, region shape and motion activity) are extracted and stored in XML format by using MPEG-7 Reference Software (eXperimentation Model - XM) [27] that provides the extraction of the low-level features of multimedia data and stores them in XML format.

The XM Software is the simulation platform for the MPEG-7 Standard and contains two types of applications: server applications and client applications. Server applications create the descriptor (D) or description scheme (DS) such as Color Layout descriptor, Dominant Color descriptor etc. By using server applications of XM, visual features of multimedia data can be automatically extracted and stored in XML files. Client applications use these extracted Ds or DSs for searching or filtering multimedia data.

Sample XML files generated by using XM software are given in the following section.

Figure 6.1 The Keyframe of a Sample Video Shot

Generated XML Document for Color Layout of the keyframe given in Figure 6.1:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
 <Mpeg7 xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance">
   <DescriptionUnit xsi:type = "DescriptorCollectionType">
      ...
      <Image name = "5.jpg">
            <Descriptor xsi:type = "ColorLayoutType">
                  <YDCCoeff>36</YDCCoeff>
                  <CbDCCoeff>13</CbDCCoeff>
                  <CrDCCoeff>38</CrDCCoeff>
                  <YACCoeff5>15  6  10  17  16 </YACCoeff5>
                  <CbACCoeff2>16  25 </CbACCoeff2>
                  <CrACCoeff2>16  10 </CrACCoeff2>
            </Descriptor>
      </Image>

      ...
   </DescriptionUnit>
 </Mpeg7>
```

The meanings of tags are:

Image tag with its *name* attribute specifies the name of the keyframe of a video shot. Color Layout descriptor or any other descriptor defined in MPEG-7 Standard does not contain this tag. Thus, this tag is added into descriptors by using the modified MPEG-7 XM version which is used in [12].

YDCCoeff and YACCoeff represent the DCT coefficients values for the Y component of YCbCr color space.

CbDCCoeff and CbACCoeff represent the DCT coefficients values for the Cb component of YCbCr color space.

CrDCCoeff and CrACCoeff represent the DCT coefficients values for the Cr component of YCbCr color space.

Generated XML Document for Dominant Color of the keyframe given in Figure 6.1:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
 <Mpeg7 xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance">
   <DescriptionUnit xsi:type = "DescriptorCollectionType">
      ...
      <Image name = "5.jpg">
        <Descriptor size = "5" xsi:type = "DominantColorType">
          <ColorSpace type = "RGB" colorReferenceFlag = "false"/>
          <ColorQuantization>
            <Component>R</Component>
            <NumOfBins>256</NumOfBins>
            <Component>G</Component>
            <NumOfBins>256</NumOfBins>
            <Component>B</Component>
            <NumOfBins>256</NumOfBins>
          </ColorQuantization>
          <SpatialCoherency>5</SpatialCoherency>
          <Values>
            <Percentage>5</Percentage>
            <ColorValueIndex>41  43  34 </ColorValueIndex>
            <ColorVariance>1  0  1 </ColorVariance>
          </Values>
          <Values>
            <Percentage>2</Percentage>
            <ColorValueIndex>195  199  181 </ColorValueIndex>
            <ColorVariance>1  0  1 </ColorVariance>
          </Values>
          <Values>
            <Percentage>18</Percentage>
            <ColorValueIndex>202  184  116 </ColorValueIndex>
            <ColorVariance>1  0  0 </ColorVariance>
          </Values>
          <Values>
            <Percentage>3</Percentage>
            <ColorValueIndex>73  92  78 </ColorValueIndex>
            <ColorVariance>1  1  0 </ColorVariance>
          </Values>
          <Values>
            <Percentage>2</Percentage>
            <ColorValueIndex>119  111  76 </ColorValueIndex>
            <ColorVariance>1  0  1 </ColorVariance>
          </Values>
        </Descriptor>
      </Image>
      ...
   </DescriptionUnit>
 </Mpeg7>
```

The meanings of tags are:

Image tag with its *name* attribute specifies the name of the keyframe of a video shot.

Size attribute in *Descriptor* tag specifies the number of dominant colors in the region.

SpatialCoherency tag specifies the spatial coherency of dominant colors of a region of interest. Examples of low and high spatial coherency are given in Figure 6.2.

(a) (b)

Figure 6.2 Examples of (a) Low and (b) High Spatial Coherency

Percentage tag specifies the percentage of pixels associated with a dominant color.

ColorVariance tag specifies the color variance of a dominant color.

ColorValueIndex tag specifies the value of dominant color in the specified color space, which is RGB for our study.

Generated XML Document for Edge Histogram of the keyframe given in Figure 6.1:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
 <Mpeg7 xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance">
   <DescriptionUnit xsi:type = "DescriptorCollectionType">
      ...
      <Image name = "5.jpg">
            <Descriptor xsi:type = "EdgeHistogramType">
                  <BinCounts>2  6  5  6  4  2  5  4  5  5  3  4  4
                  5  5  4  3  2  6  5  5  4  6  5  1  3  4  5  5  4
                  1  5  3  6  4  5  4  4  2  2  0  6  3  1  3  2  5
                  3  2  5  2  4  4  4  5  2  5  2  5  5  0  6  4  5
                  4  4  2  6  4  5  0  5  2  2  4  2  6  4  5  4
                  </BinCounts>
            </Descriptor>
      </Image>

      ...
   </DescriptionUnit>
 </Mpeg7>
```

As explained in Section 2.1.3.2, Edge Histogram descriptor categorizes the edges of a region of interest (i.e. image or keyframe) into five types. The region of interest is divided

into 16 sub-images and the number of edge types is counted for each sub-image that result 80 bincounts as shown in the sample Edge Histogram descriptor above.

Generated XML Document for Region Shape of the keyframe given in Figure 6.1:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
 <Mpeg7 xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance">
   <DescriptionUnit xsi:type = "DescriptorCollectionType">
      ...
      <Image name = "5.jpg">
            <Descriptor xsi:type = "RegionShapeType">
                  <MagnitudeOfART>14  15  13  14  8  15  11  15  15
                  7  15  13  10  14  15  11  14  12  4  11  15  12
                  11  8  4  12  4  2  7  12  5  11  13  8  11
                  </MagnitudeOfART>
            </Descriptor>
      </Image>
      ...
   </DescriptionUnit>
 </Mpeg7>
```

MagnitudeOfART tag specifies the magnitude of 35 Angular Radial Transform (ART coefficients that describe the shape.

Generated XML Document for Motion Activity of the sample video shot whose keyframe is given in Figure 6.1:

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
 <Mpeg7 xmlns:xsi = "http://www.w3.org/2000/10/XMLSchema-instance">
   <DescriptionUnit xsi:type = "DescriptorCollectionType">
      ...
      <Image name = "5.mpg">
            <Descriptor xsi:type="MotionActivity">
                  <Intensity>4</Intensity>
                  <DominantDirection>0</DominantDirection>
                  <SpatialParameters>
                        <Nsr>38</Nsr>
                        <Nmr>4</Nmr>
                        <Nlr>1</Nlr>
                  </SpatialParameters>
                  <SpaLocNumber/>
                  <SpatialLocalizationParameters/>
                  <TemporalParameters>7 8 7 31 8
                  </TemporalParameters>
            </Descriptor>
      </Image>
      ...
   </DescriptionUnit>
 </Mpeg7>
```

Detailed explanation of Motion Activity descriptor is given in Section 2.1.3.4.

6.3 Raw Data and Feature Storage

As stated in Chapter 3, the video shots and their related keyframes are stored in a pre-defined location on the file system for further processing (i.e. low-level feature extraction). Video shots are stored in mpg format, where their related keyframes are stored in jpg format.

We store extracted low-level features of video shots in the XML database, namely Oracle Berkeley DB [28]. In Oracle Berkeley DB, CL, DC, EH and RS features of the keyframes of video shots and MA features of video shots are stored, within a seperate XML file for each visual feature. Oracle Berkeley DB is an open source native XML database that provides Java and C++ APIs for the integration of applications with a fast, reliable and scalable XML database. By using Java API of the XML database, we make XPath queries over Oracle Berkeley DB to get the visual features of video shots stored in the database.

6.4 Indexing

In our content-based retrieval system, Slim-Tree is implemented by using XXL (eXtensible and fleXible Library for data processing) API [29]. XXL API is a high-level, extensible and flexible, platform independent Java API for data processing and database purposes. XXL API provides a collection of easy-to-use index structures and query operators facilitating the performance evaluation of index structures.

To construct Slim-Tree, we provide the distance functions given in Section 3.5.1 and the feature values (i.e. color layout, dominant color, edge histogram, region shape and motion activity descriptors) of the video shots to XXL API. Color layout, dominant color, region shape, edge histogram and motion activity low-level descriptors of video shots are fetched from the XML database using XPath queries. Then using XXL API, Slim-Tree is constructed with Minimal Spanning Tree (MST) split strategy and Mindist sub-tree choosing strategy. The details of MST split strategy and Mindist sub-tree choosing strategy is discussed in Chapter 4.

For BitMatrix implementation, Weka API [30] is used to cluster the low-level visual features of video shots and Colt API [31] is used to hold the constructed BitMatrix and to make operations on the BitMatrix such as taking the intersection of two bitmap signatures during similarity queries. Algorithms for BitMatrix construction/update is given in Chapter

5. Thus, here we briefly explain how we construct BitMatrix index structure for video shots.

To construct BitMatrix, firstly color layout, dominant color, region shape, edge histogram and motion activity low-level descriptors of video shots are fetched from the XML database using XPath queries. After fetching low-level descriptors from the XML database, an Attribute-Relation File Format (ARFF) file is created for each feature (i.e. CL, DC, EH, RS and MA), seperately. The ARFF files are used by the Weka API during clustering, and they are the files that define the attributes of the video data to be clustered and that contain the related feature values of the video shots. Sample ARFF file for the CL feature is given in the following section to clarify the content of an ARFF file.

```
@relation ColorLayout

@attribute YDCCoeff numeric
@attribute CbDCCoeff numeric
@attribute CrDCCoeff numeric
@attribute YACCoeff1 numeric
@attribute YACCoeff2 numeric
@attribute YACCoeff3 numeric
@attribute YACCoeff4 numeric
@attribute YACCoeff5 numeric
@attribute CbACCoeff1 numeric
@attribute CbACCoeff2 numeric
@attribute CrACCoeff1 numeric
@attribute CrACCoeff2 numeric

@data
17,22,40,12,22,14,15,18,16,14,21,16
...
```

As seen from the sample ARFF file, first the CL attributes to be used in clustering are defined, then the CL feature values of the video shots are given. The attributes defined in the ARFF file are the attributes of features to be used in the distance computation of video shots.

In this study, since the nature of the feature values (i.e. the defined range of the feature values) are known, *k-means clustering* algorithm of the Weka API is used as the clustering algorithm. The k-means algorithm clusters n objects based on attributes into k partitions, where $k < n$. The aim of the algorithm is to minimize total intra-cluster variance. In this study, after the construction of the input ARFF files for clustering, video shots are clustered using k-means algorithm into pre-defined number of clusters according to their CL, DC, EH, RS and MA values, seperately. After this clustering step, for each video shot the

bitmap signature is constructed by using the clustering results of CL, DC, EH, RS and MA features. Finally, the resulting BitMatrix that contains all the bitmap signatures for video shots in the data set is built using these bitmap signatures.

6.5 Querying Module

As stated in Chapter 3, the system uses two index structures, namely Slim-Tree and BitMatrix for content-based retrieval. Content-based retrieval methods implemented for Slim-Tree can be summarized as follows. Low-level feature descriptors (i.e. Color Layout, Dominant Color, Edge Histogram, Region Shape and Motion Activity) of the query video shot are fetched from the XML database using XPath queries. Then, the query methods defined in XXL API are used to compare the low-level features of the query video shot with the video shots in the current Slim-Tree by computing distance values as defined in Section 3.5. The input to the XXL API is the type of query (exact match, k-NN or range query), feature values of the query video shot, and distance functions and OWA weights to be used. Retrieval results are returned according to query type.

Content-based retrieval methods implemented for BitMatrix can be summarized as follows. Low-level feature descriptors (i.e. Color Layout, Dominant Color, Edge Histogram, Region Shape and Motion Activity) of the query video shot are fetched from the XML database using XPath queries. Then, by using Weka API the bitmap signature of the query video shot is generated with the fetched low-level features. The bitmap signature of the query video shot is bitwise ANDed with bitmap signatures of all video shots on the current BitMatrix. If the cardinality threshold is above 2 (i.e. two compared video shots are in the same range of at least two feature dimensions), then the video shot on the current BitMatrix is retained for further processing. Visual feature values (i.e. Color Layout, Dominant Color, Edge Histogram, Region Shape and Motion Activity) of the remaining video shots are accessed and exact distances between the query video shot and video shots on the current BitMatrix are computed as defined in Section 3.5, sequentially. Retrieval results are returned according to query type.

For both index structures, if the query is an exact match/range query then the video shots which have less distance value than range radius that is zero for exact match are returned. Otherwise, the query is a k-NN query and the most k similar video shots to the query video shot are ranked and returned.

CHAPTER 7

PERFORMANCE TESTS

Video shots from MPEG-7 test set and random video shots from Web were used to test the performance of our content-based video retrieval system. Index construction tests were done over video databases that contain 100, 300, 500, 700 and 1000 video shots to evaluate the number of distance computations and construction times of the index structures. Tests on insert/delete operations of index structures were done by using 100, 200, 300, 400 and 500 video shots and the number of distance computations and insertion/deletion times were evaluated. Our system also was tested by k-NN and range query paradigm. With these tests, the number of distance computations and query response times were examined. In addition, retrieval efficiency of the system was evaluated by using Average Normalized Modified Retrieval Rank (ANMRR) metric [4] and precision/recall values. We also applied some tests on Corel Database [46] images similar to the tests done on video data to evaluate the performance of Slim-Tree and BitMatrix index structures on images.

7.1 Building the Index Structures

In this section, the performance of building BitMatrix and Slim-Tree index structures are evaluated over images and videos, respectively.

7.1.1 Building the Index Structures for Images

For image databases, BitMatrix and Slim-Tree index structures were constructed by using two different approaches: one is giving equal weights to each image feature (i.e. color, texture and shape) during distance computation of any two images, and the other is using OWA operators to give weights to each image feature in distance computation of any two images. Used weights of OWA operator for images are {0.4, 0.3, 0.2, 0.1} and these weights are used throughout the whole performance tests on images.

Tests were performed for three different minimum utilization values (only applicable to Slim-Tree) and four different databases were used. The results are shown in the following figures.

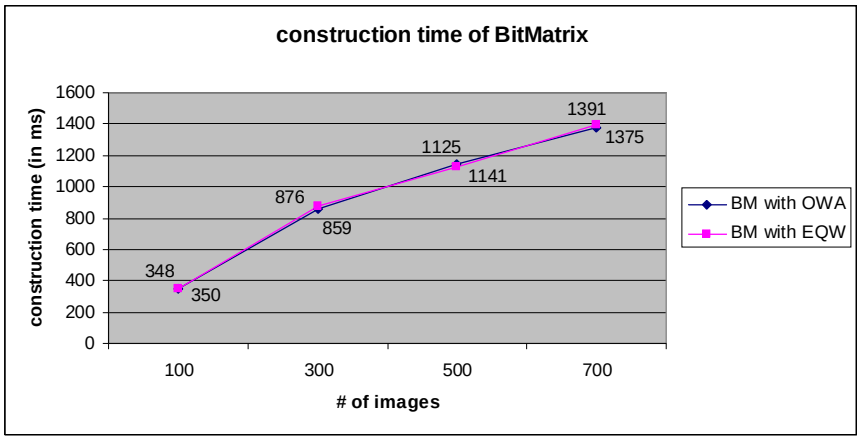

Figure 7.1 Construction Time of BitMatrix for Images as a Function of # of Images

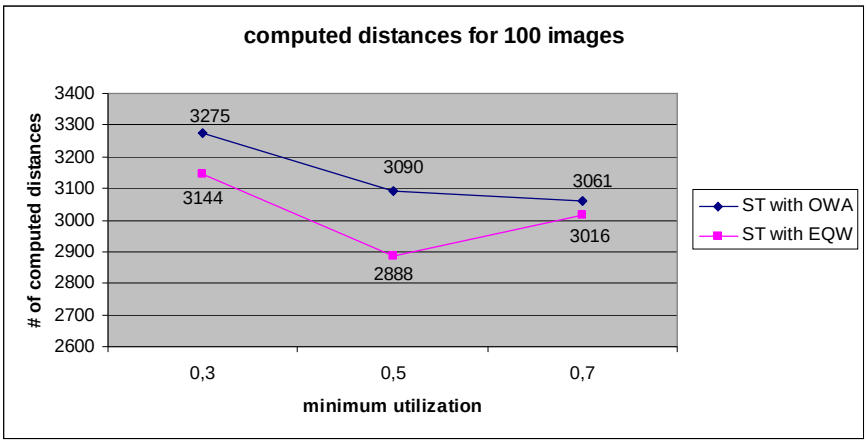

Figure 7.2 # of Computed Distances for 100 Images as a Function of Minimum Utilization

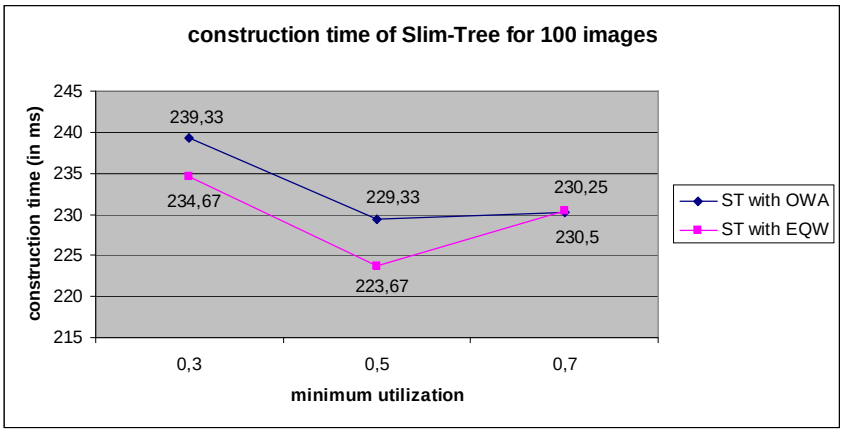

Figure 7.3 Construction Time of Slim-Tree for 100 Images as a Function of Minimum Utilization

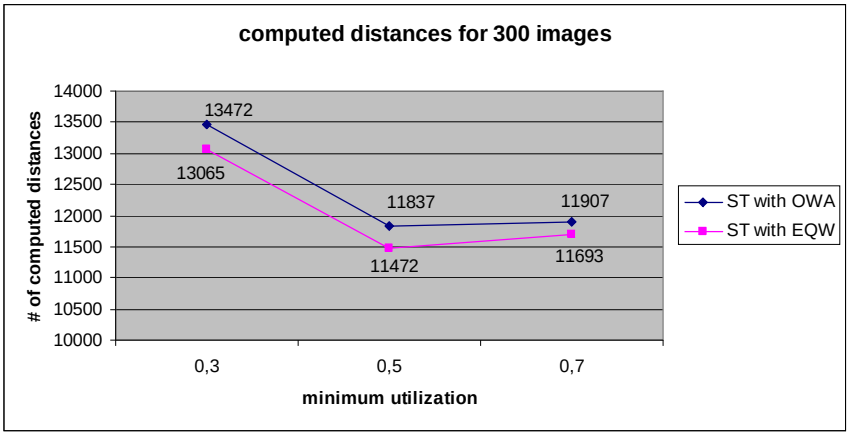

Figure 7.4 # of Computed Distances for 300 Images as a Function of Minimum Utilization

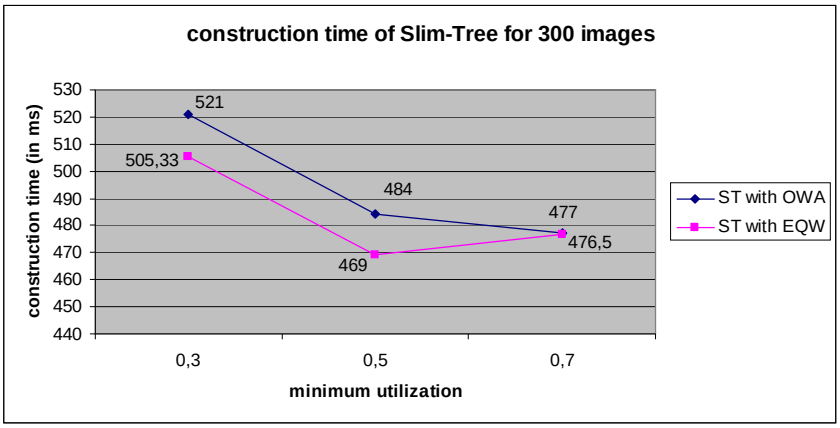

Figure 7.5 Construction Time of Slim-Tree for 300 Images as a Function of Minimum Utilization

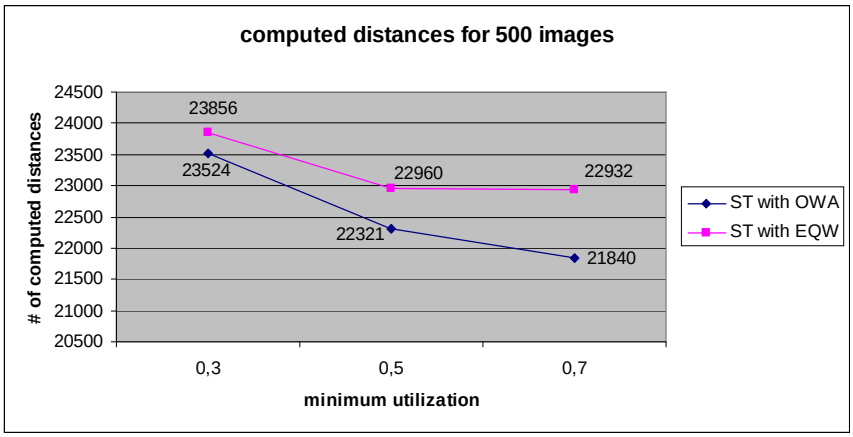

Figure 7.6 # of Computed Distances for 500 Images as a Function of Minimum Utilization

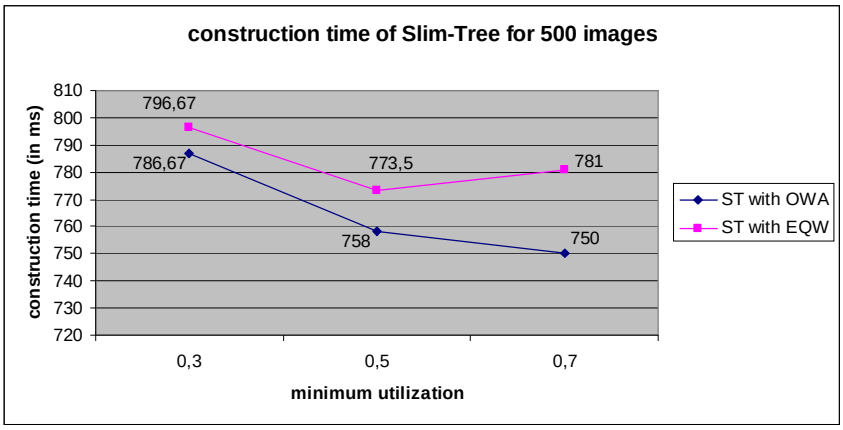

Figure 7.7 Construction Time of Slim-Tree for 500 Images as a Function of Minimum Utilization

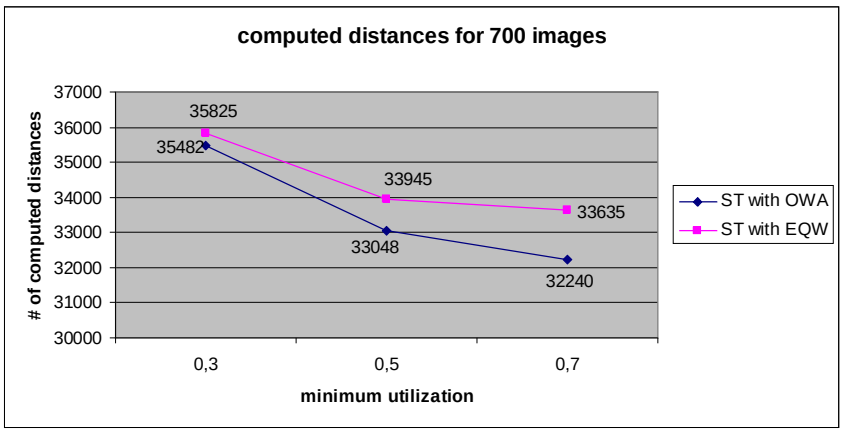

Figure 7.8 # of Computed Distances for 700 Images as a Function of Minimum Utilization

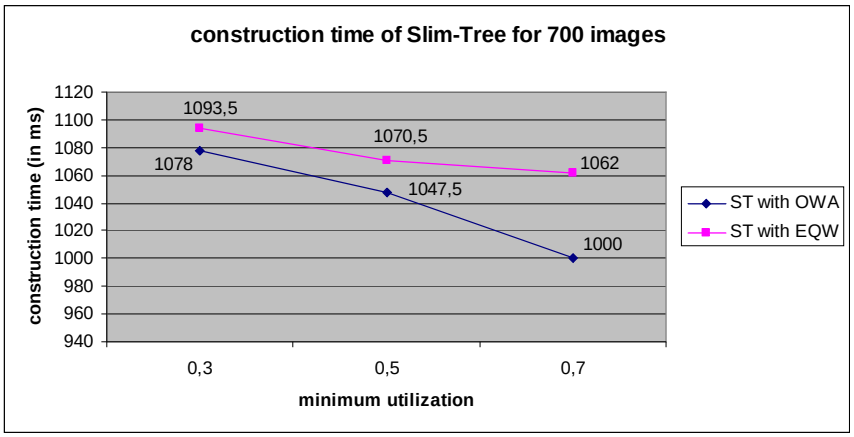

Figure 7.9 Construction Time of Slim-Tree for 700 Images as a Function of Minimum Utilization

As seen from the index construction test results of images (from Figure 7.1 to Figure 7.9), we can say that BitMatrix index construction time is worse than Slim-Tree's because of the clustering process during BitMatrix index construction and this clustering time increases as the number of images is increased as expected. Using OWA operator does not affect the BitMatrix index construction performance, since there is no distance computation during BitMatrix construction. In addition, Slim-Tree with OWA performs better than Slim-Tree with EQW in terms of number of distance computations and elapsed time for index construction as the number of images increase.

7.1.2 Building the Index Structures for Videos

For videos, BitMatrix and Slim-Tree index structures were constructed by using two different approaches as for images. First approach gives equal weights to each video feature (i.e. color, texture, shape and motion) during distance computation of any two video shots; and the other approach uses OWA operators to give weights to each feature of a video shot in distance computation of any two video shots. Used weights of OWA operator for videos are {0.3, 0.3, 0.2, 0.1, 0.1} and these weights are used throughout the whole performance tests on videos.

Tests were performed for three different minimum utilization values (only applicable to Slim-Tree) and five different databases were used. The results are shown in the following figures.

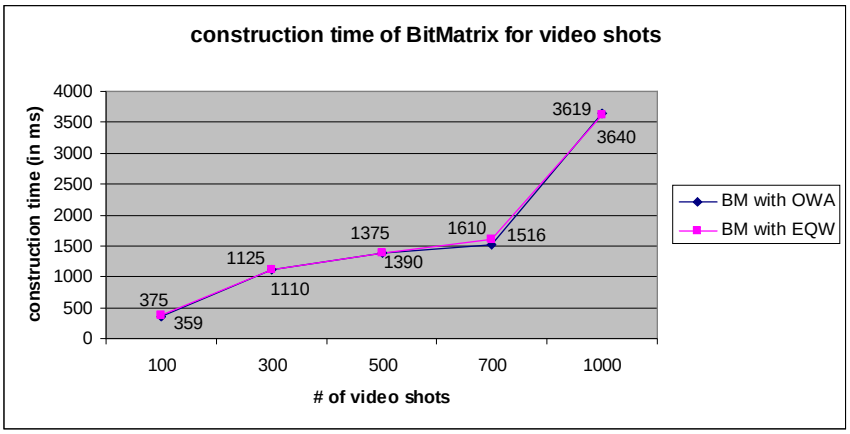

Figure 7.10 Construction Time of BitMatrix for Video Shots as a Function of # of Video Shots

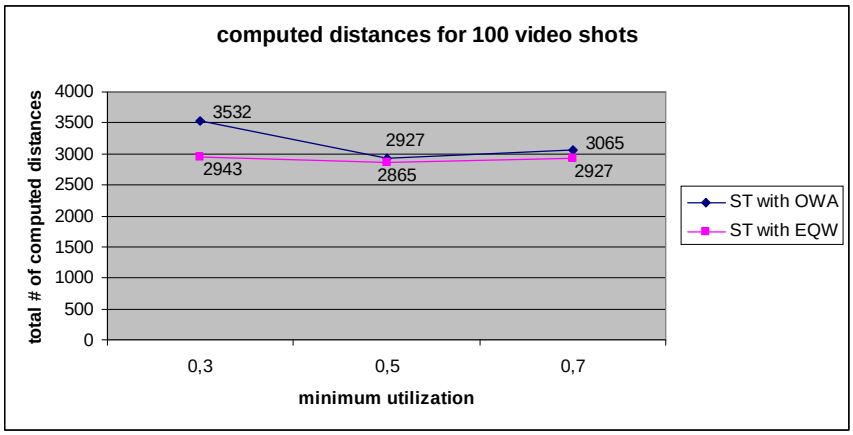

Figure 7.11 # of Computed Distances for 100 Video Shots as a Function of Minimum Utilization

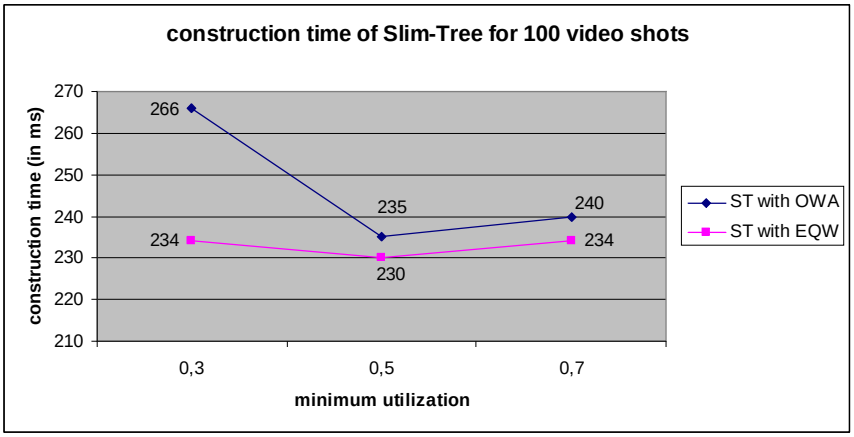

Figure 7.12 Construction Time of Slim-Tree for 100 Video Shots as a Function of Minimum Utilization

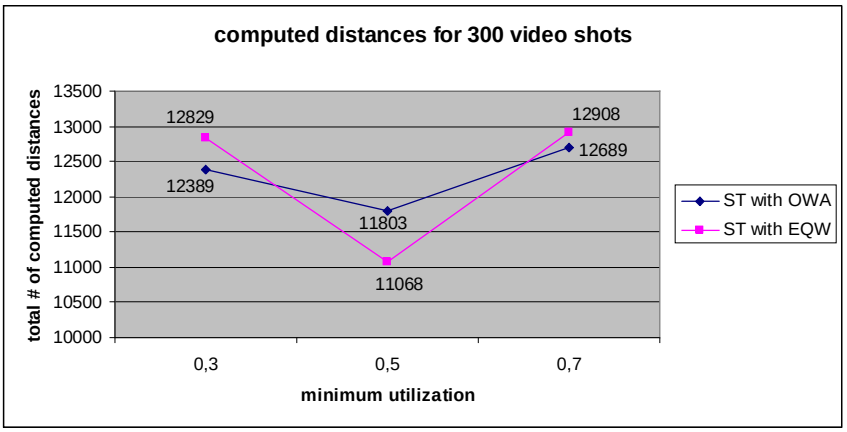

Figure 7.13 # of Computed Distances for 300 Video Shots as a Function of Minimum Utilization

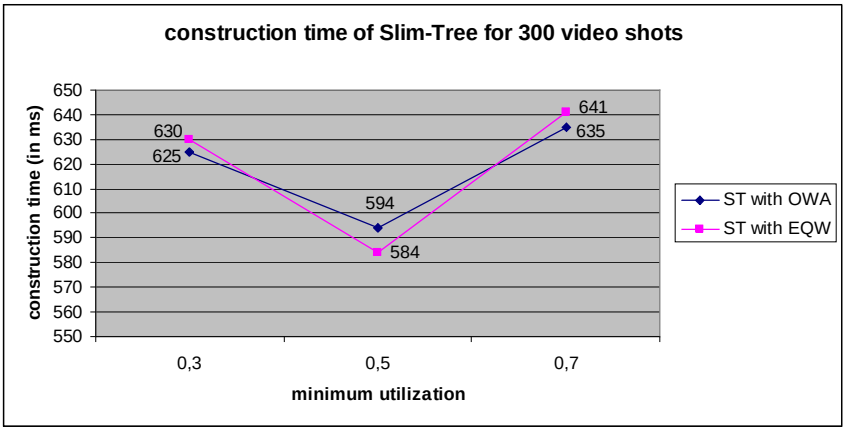

Figure 7.14 Construction Time of Slim-Tree for 300 Video Shots as a Function of Minimum Utilization

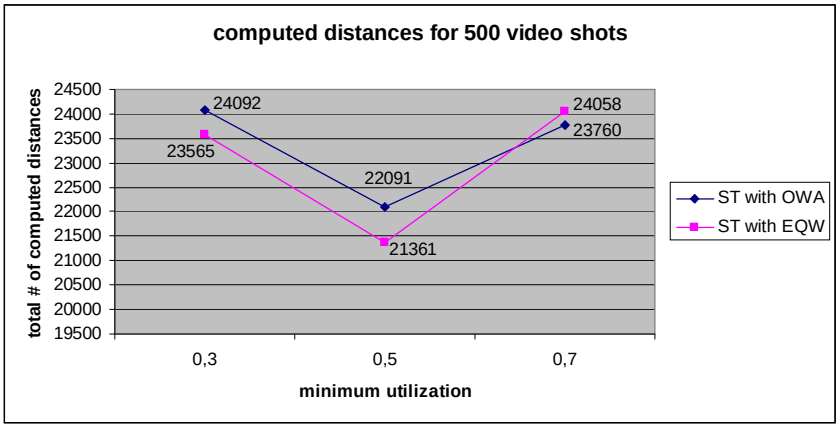

Figure 7.15 # of Computed Distances for 500 Video Shots as a Function of Minimum Utilization

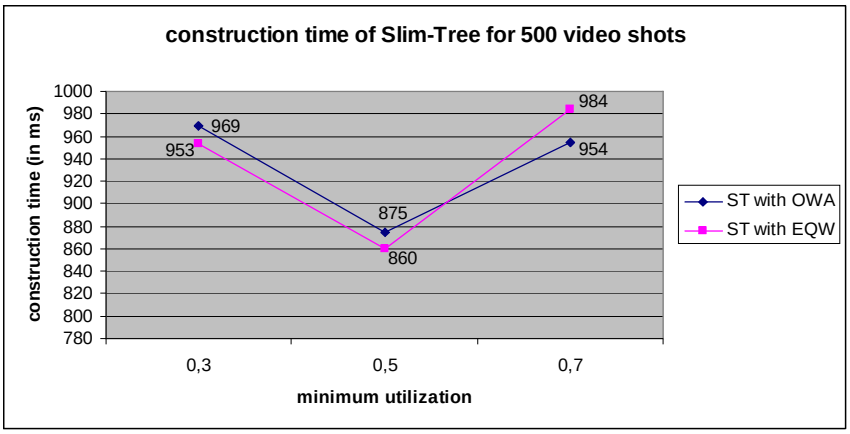

Figure 7.16 Construction Time of Slim-Tree for 500 Video Shots as a Function of Minimum Utilization

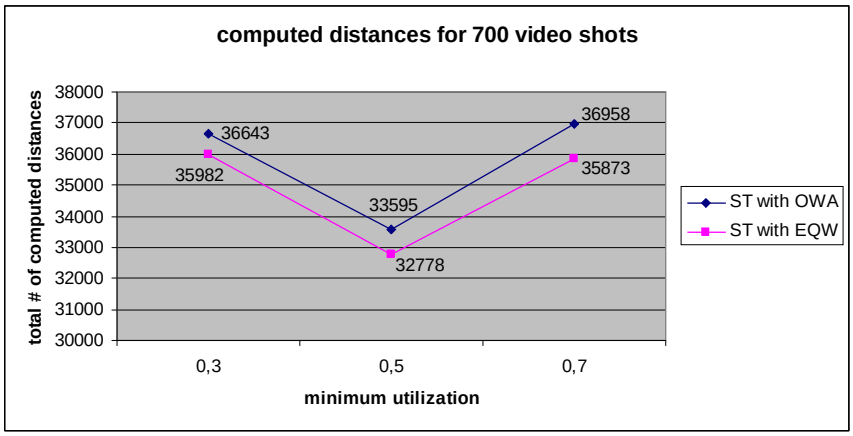

Figure 7.17 # of Computed Distances for 700 Video Shots as a Function of Minimum Utilization

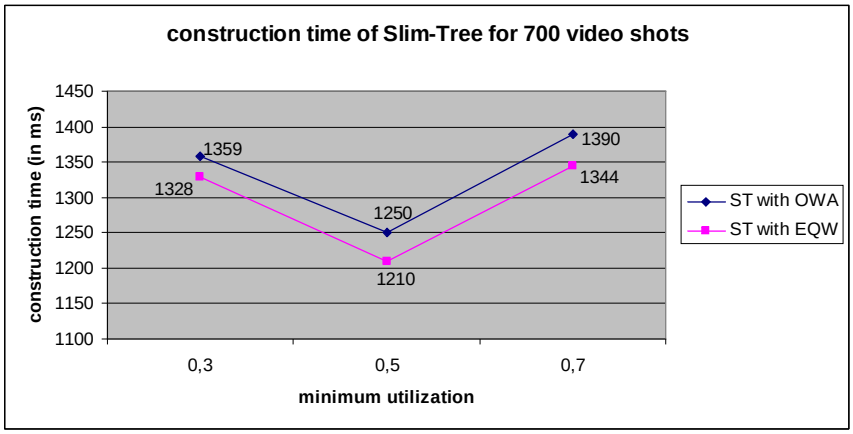

Figure 7.18 Construction Time of Slim-Tree for 700 Video Shots as a Function of Minimum Utilization

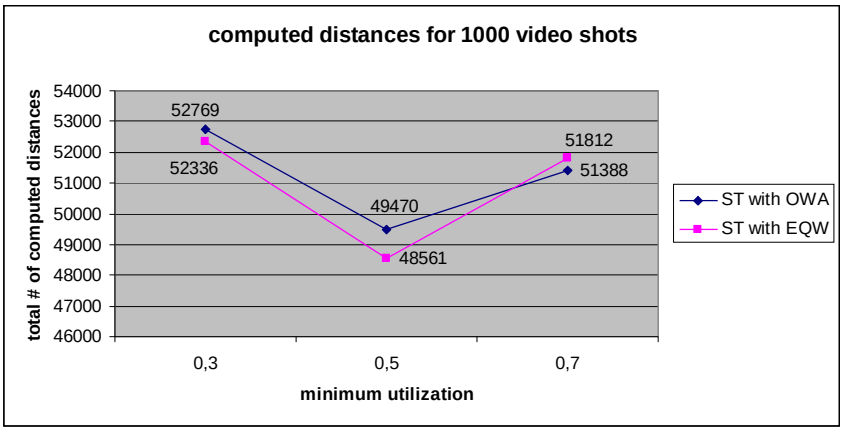

Figure 7.19 # of Computed Distances for 1000 Video Shots as a Function of Minimum Utilization

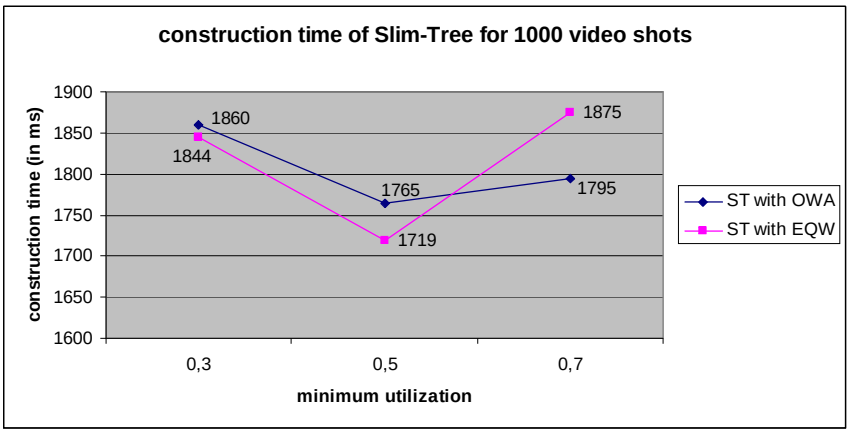

Figure 7.20 Construction Time of Slim-Tree for 1000 Video Shots as a Function of Minimum Utilization

As seen from the index construction test results of video shots (from Figure 7.10 to Figure 7.20), as in images we can say that BitMatrix index construction time is worse than Slim-Tree because of the clustering process during BitMatrix index construction and this clustering time increases as the number of video shots is increased as expected. Also, using OWA operator does not affect the BitMatrix index construction performance, since there is no distance computation during BitMatrix construction. In addition, Slim-Tree with OWA generally performs better than Slim-Tree with EQW in terms of number of distance computations and elapsed time for index construction with high minimum utilization values such as 0.7. The index construction test results of video shots for Slim-Tree are different from the index construction test results of images, since for video shots in addition to color,

texture and shape features, motion feature is indexed by Slim-Tree and different weights are used in OWA operator.

7.2 Updating the Index Structures

In this section, update performances of BitMatrix and Slim-Tree index structures are evaluated over images and videos, respectively.

7.2.1 Updating the Index Structures for Images

Tests were performed for three different minimum utilization values (only applicable to Slim-Tree) and a database of 1000 Corel images was used. The results are shown in the following figures.

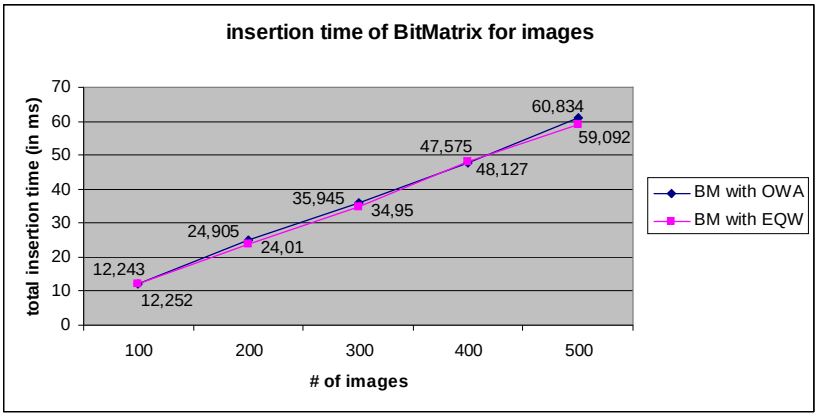

Figure 7.21 Total Insertion Time of BitMatrix for Images as a Function of # of Images

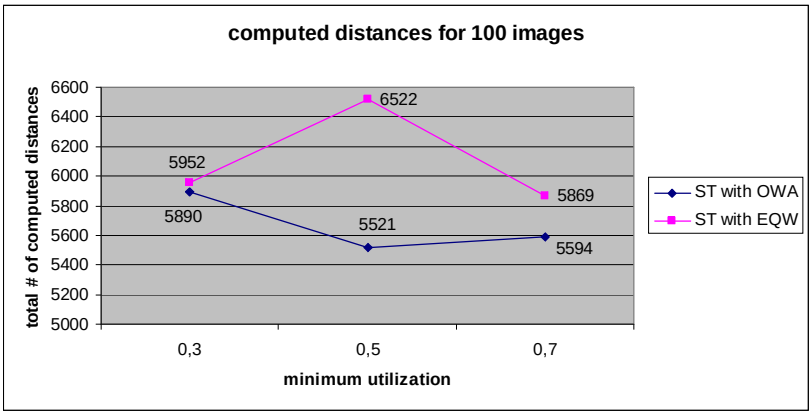

Figure 7.22 # of Computed Distances for 100 Image Insertions as a Function of Minimum Utilization

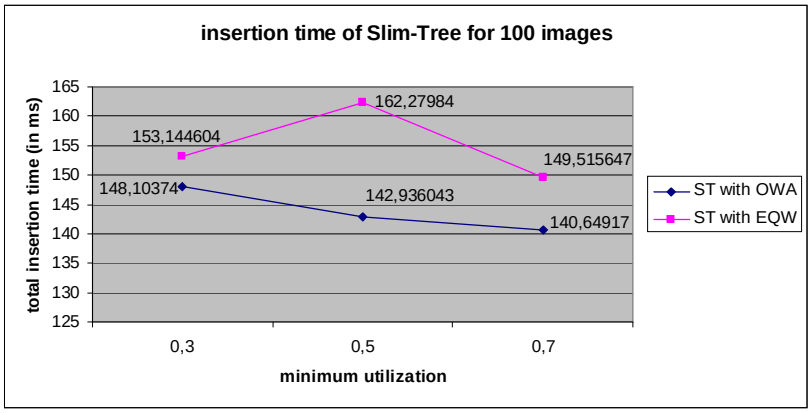

Figure 7.23 Total Insertion Time of Slim-Tree for 100 Images as a Function of Minimum Utilization

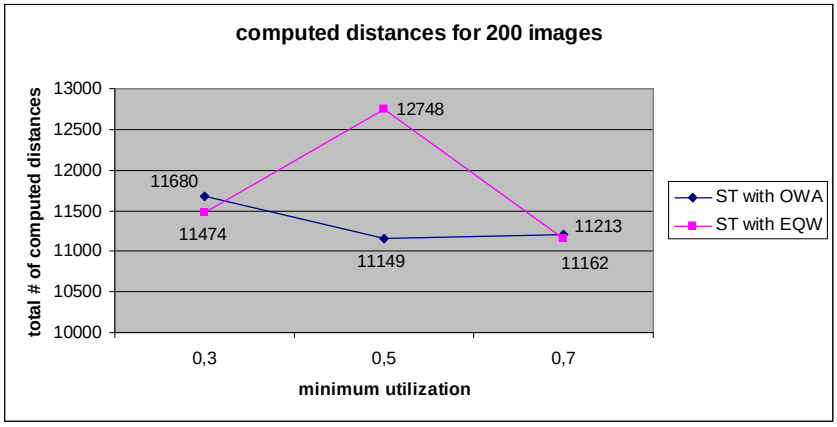

Figure 7.24 # of Computed Distances for 200 Image Insertions as a Function of Minimum Utilization

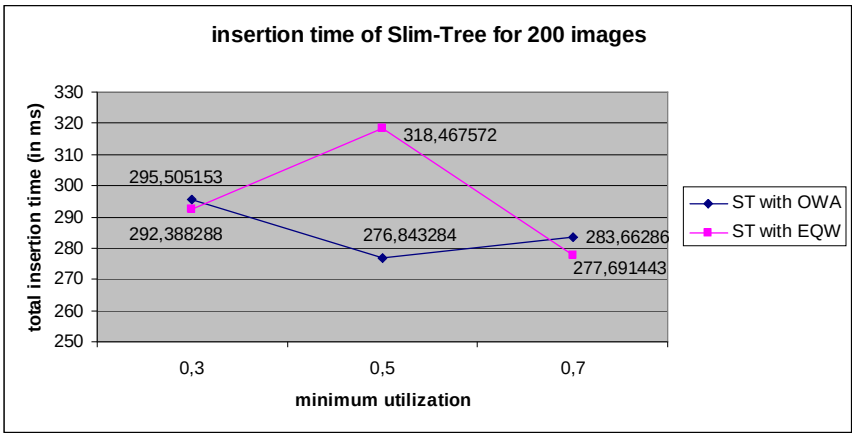

Figure 7.25 Total Insertion Time of Slim-Tree for 200 Images as a Function of Minimum Utilization

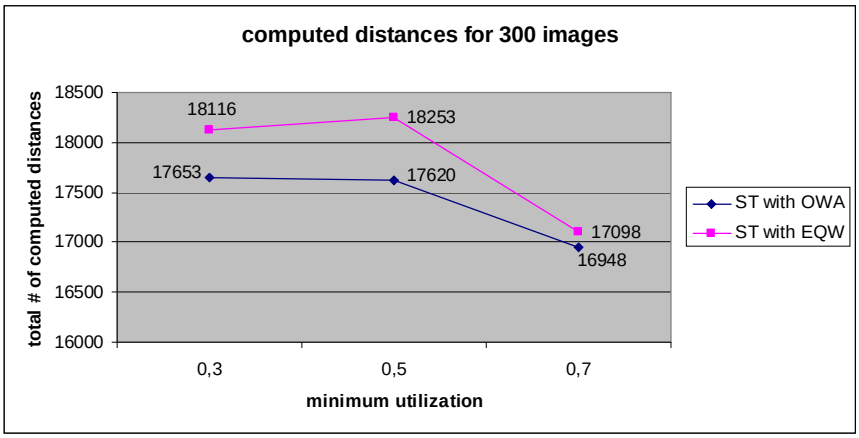

Figure 7.26 # of Computed Distances for 300 Image Insertions as a Function of Minimum Utilization

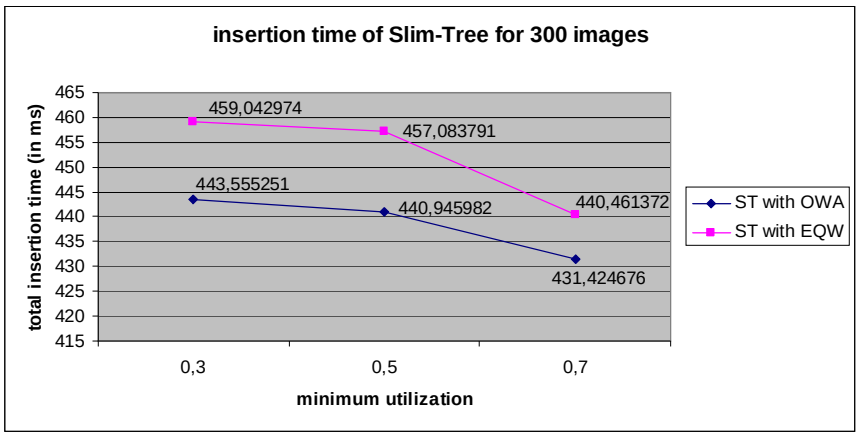

Figure 7.27 Total Insertion Time of Slim-Tree for 300 Images as a Function of Minimum Utilization

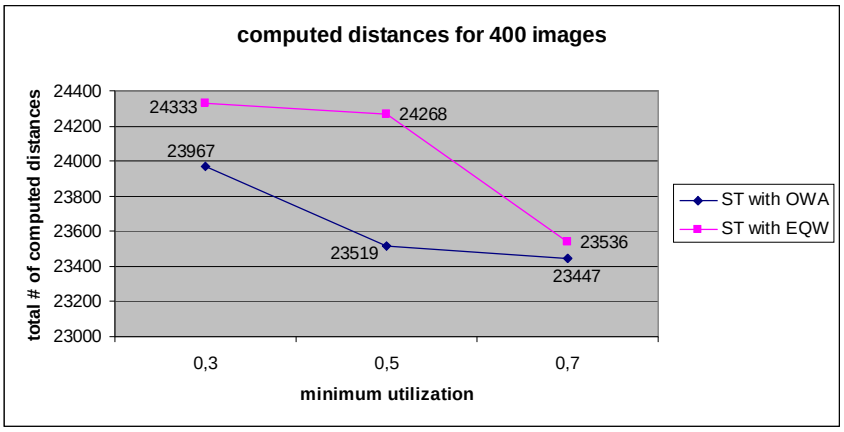

Figure 7.28 # of Computed Distances for 400 Image Insertions as a Function of Minimum Utilization

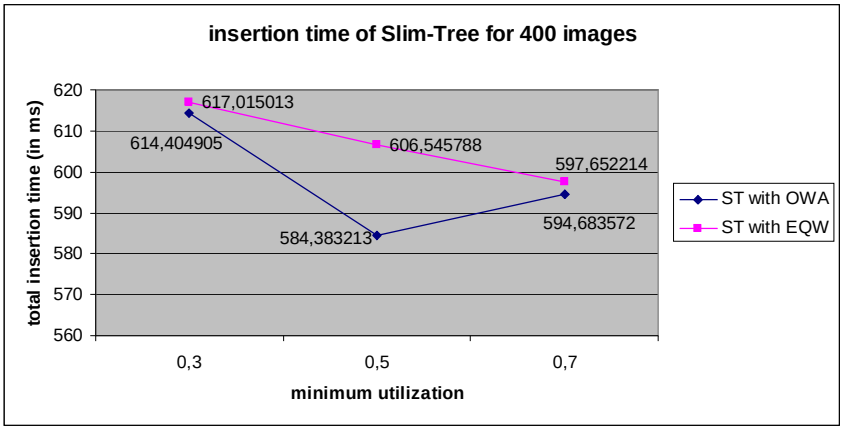

Figure 7.29 Total Insertion Time of Slim-Tree for 400 Images as a Function of Minimum Utilization

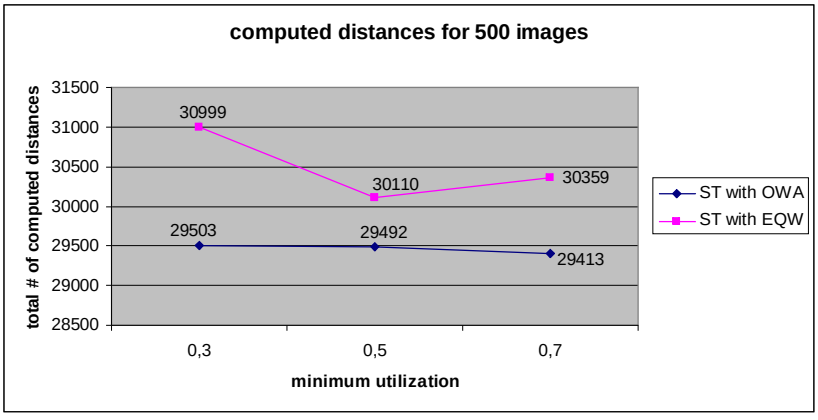

Figure 7.30 # of Computed Distances for 500 Image Insertions as a Function of Minimum Utilization

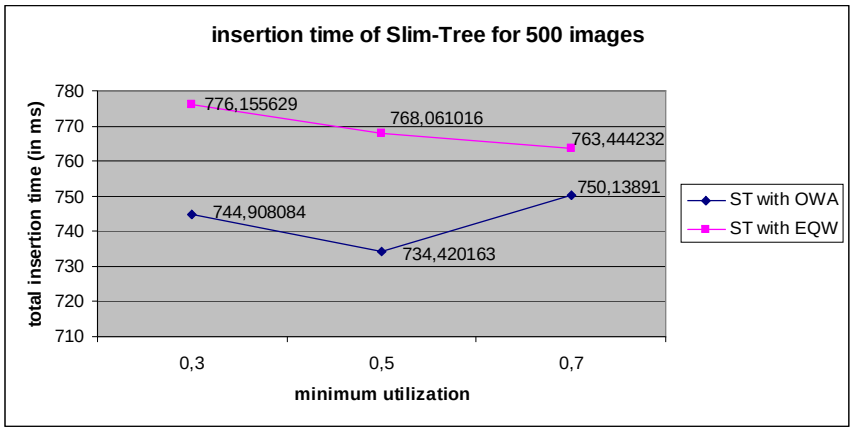

Figure 7.31 Total Insertion Time of Slim-Tree for 500 Images as a Function of Minimum Utilization

As seen from the insertion test results of index structures (from Figure 7.21 to Figure 7.31), insertion performance of BitMatrix is better than Slim-Tree using OWA or EQW. The reason for this result is doing no distance computation during insertion into BitMatrix, only low-level features of the image to be inserted are clustered to generate the bitmap signature and the related bitmap signature is inserted into BitMatrix. As in BitMatrix construction, the BitMatrix insertion time increases as the number of processed objects is increased and using OWA or EQW does not affect the insertion times as expected. In addition, for images the insertion performance of Slim-Tree with OWA is better than Slim-Tree with EQW in terms of the number of distance computations and elapsed times.

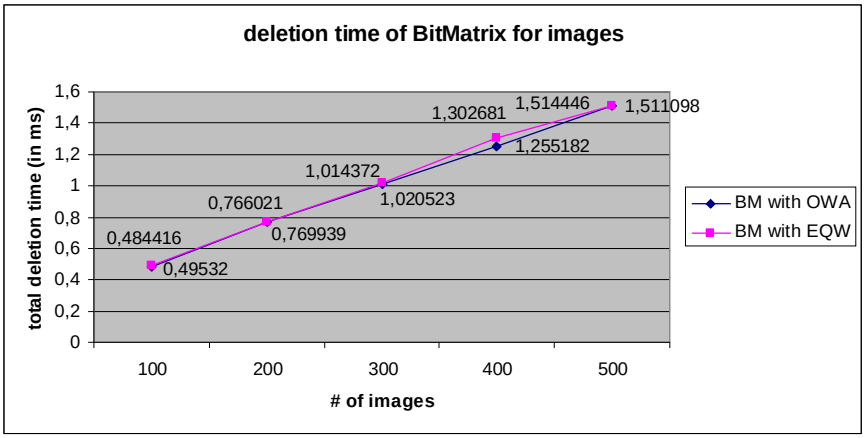

Figure 7.32 Total Deletion Time of BitMatrix for Image Deletions as a Function of # of Images

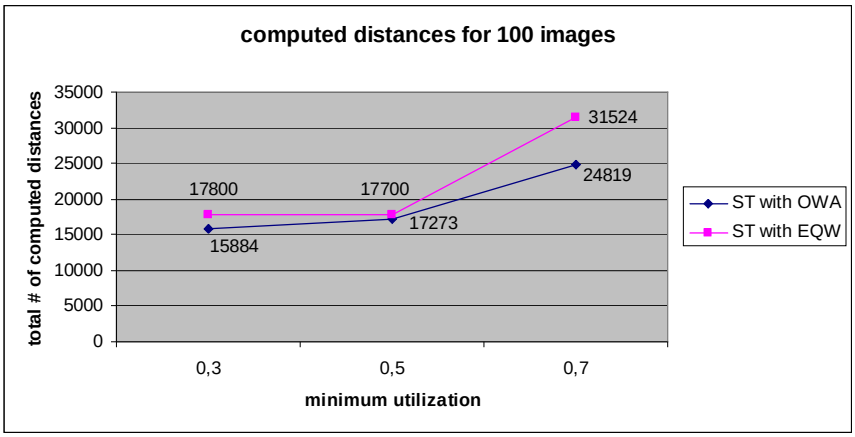

Figure 7.33 # of Computed Distances for 100 Image Deletions as a Function of Minimum Utilization

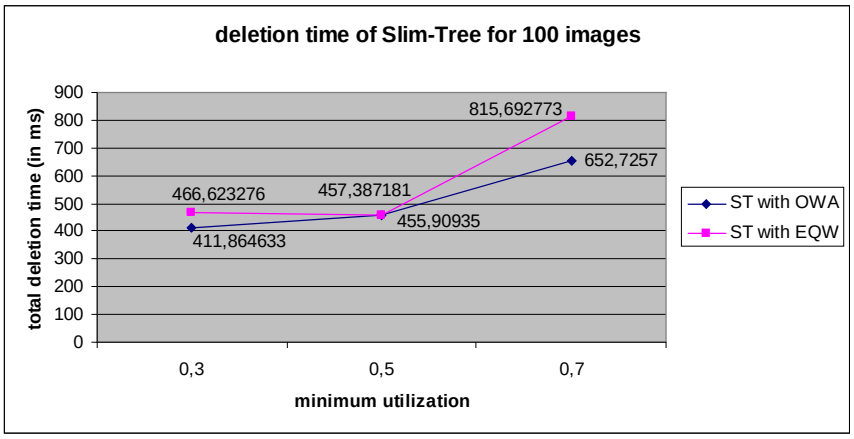

Figure 7.34 Total Deletion Time of Slim-Tree for 100 Images as a Function of Minimum Utilization

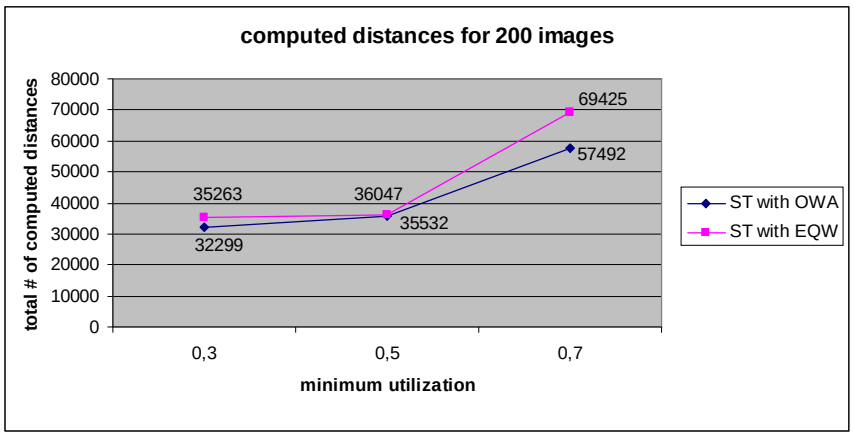

Figure 7.35 # of Computed Distances for 200 Image Deletions as a Function of Minimum Utilization

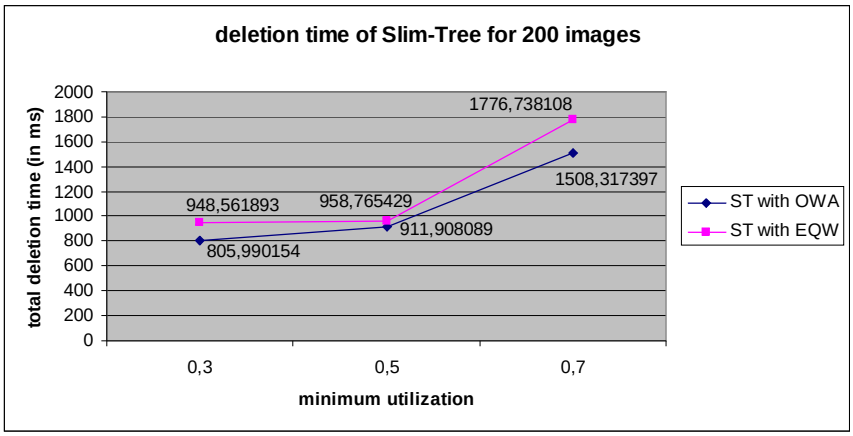

Figure 7.36 Total Deletion Time of Slim-Tree for 200 Images as a Function of Minimum Utilization

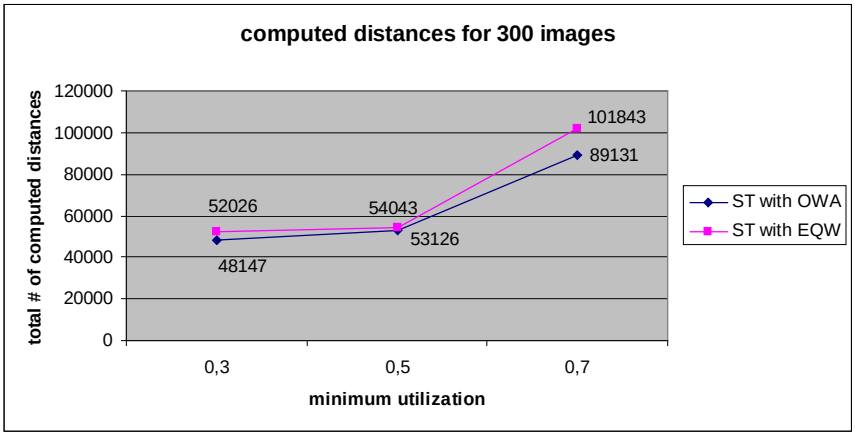

Figure 7.37 # of Computed Distances for 300 Image Deletions as a Function of Minimum Utilization

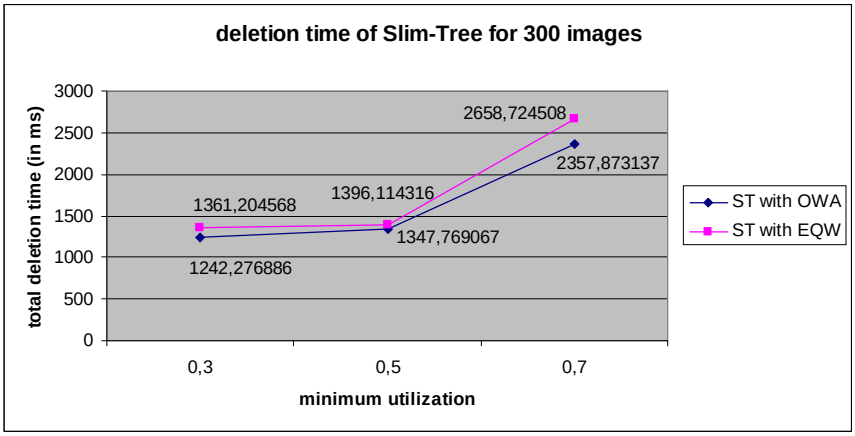

Figure 7.38 Total Deletion Time of Slim-Tree for 300 Images as a Function of Minimum Utilization

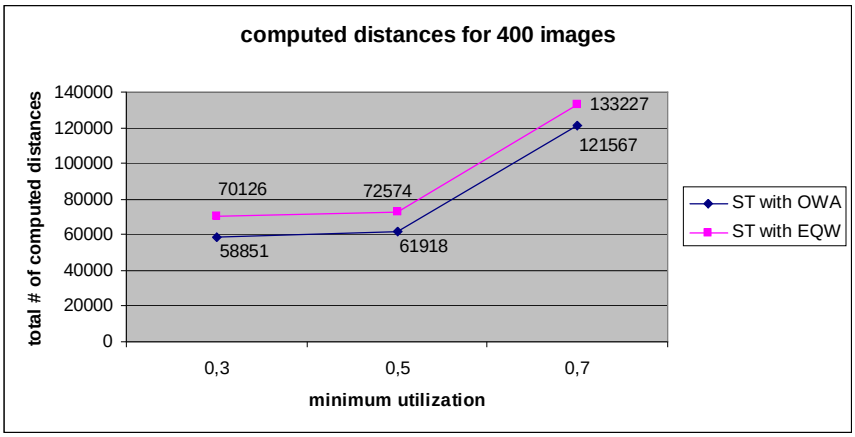

Figure 7.39 # of Computed Distances for 400 Image Deletions as a Function of Minimum Utilization

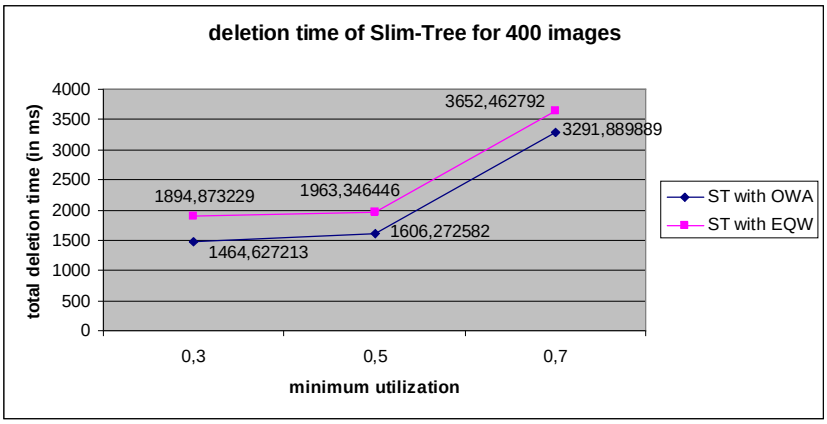

Figure 7.40 Total Deletion Time of Slim-Tree for 400 Images as a Function of Minimum Utilization

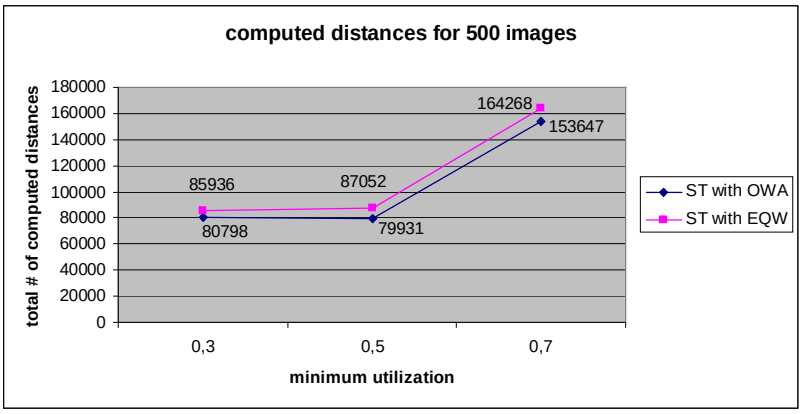

Figure 7.41 # of Computed Distances for 500 Image Deletions as a Function of Minimum Utilization

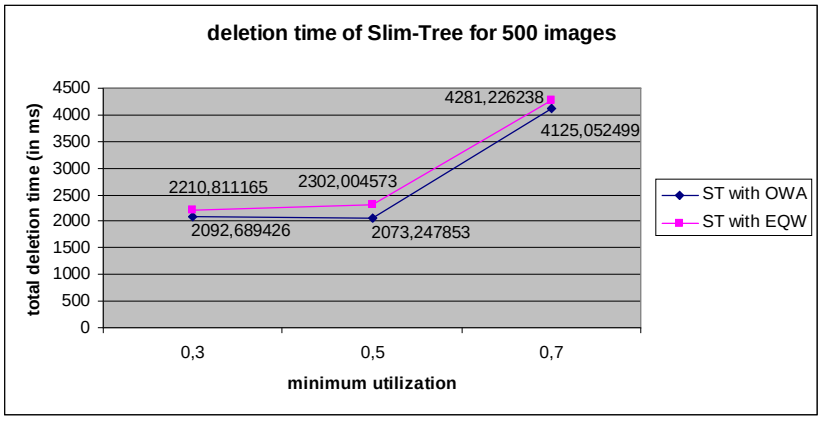

Figure 7.42 Total Deletion Time of Slim-Tree for 500 Images as a Function of Minimum Utilization

As seen from the deletion test results of index structures (from Figure 7.32 to Figure 7.42), deletion performance of BitMatrix is much better than Slim-Tree using OWA or EQW. The reason for this result is doing no distance computation during deletion, only related bitmap signature is deleted from BitMatrix. As in BitMatrix construction, the BitMatrix deletion time increases as the number of processed objects is increased and using OWA or EQW does not affect the deletion times as expected. In addition, for images the deletion performance of Slim-Tree with OWA is better than Slim-Tree with EQW in terms of the number of distance computations and elapsed times.

7.2.2 Updating the Index Structures for Videos

Tests were performed for three different minimum utilization values (only applicable to Slim-Tree) and a database of 1000 video shots from MPEG-7 test set and random video shots from Web was used. The results are shown in the following figures.

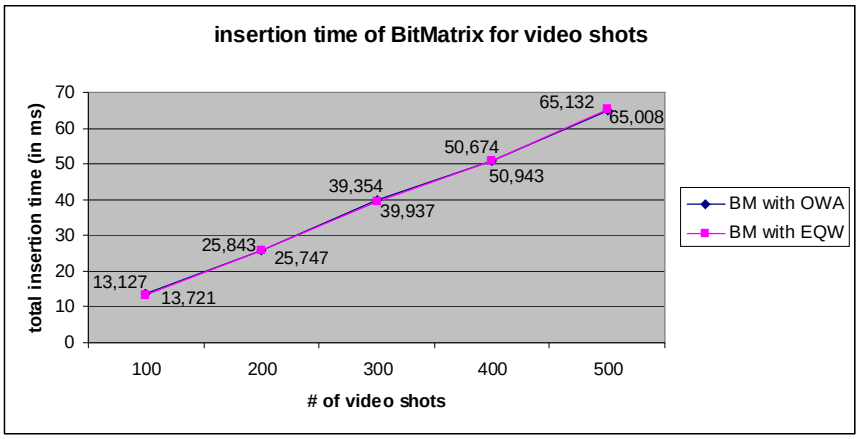

Figure 7.43 Total Insertion Time of BitMatrix for Video Shots as a Function of # of Video Shots

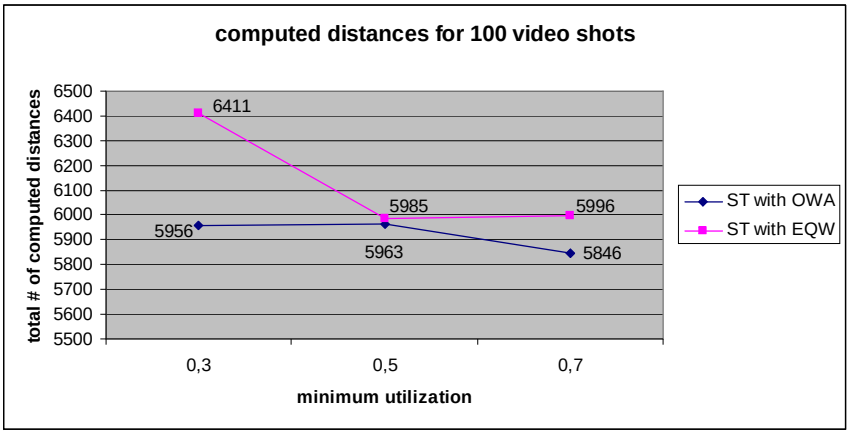

Figure 7.44 # of Computed Distances for 100 Video Shot Insertions as a Function of Minimum Utilization

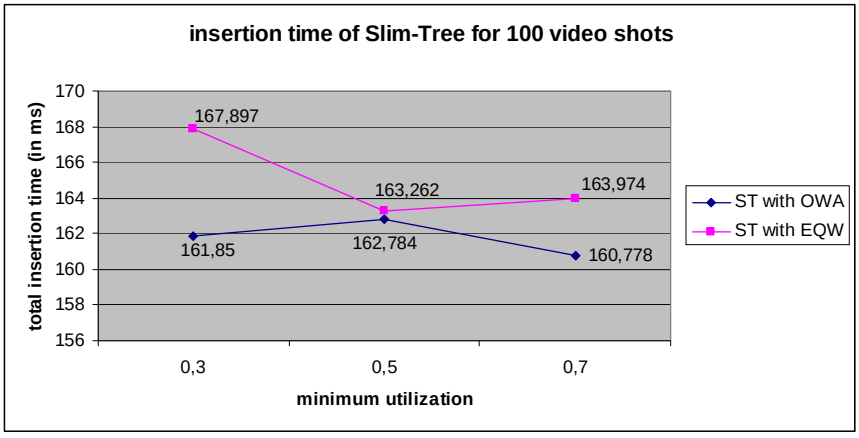

Figure 7.45 Total Insertion Time of Slim-Tree for 100 Video Shots as a Function of Minimum Utilization

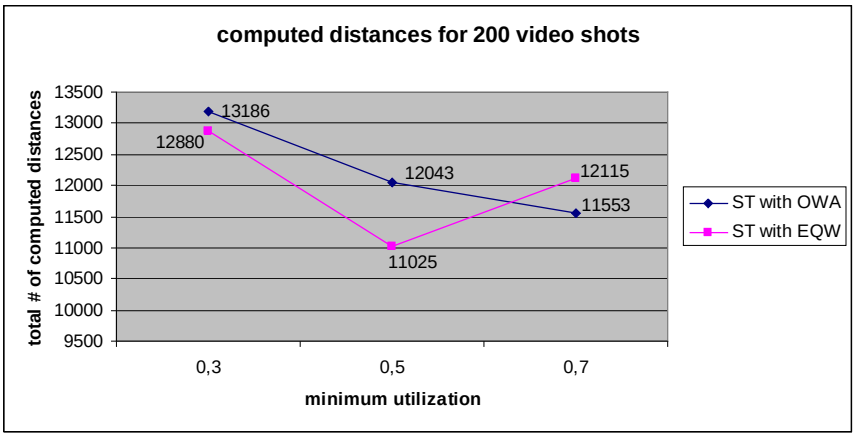

Figure 7.46 # of Computed Distances for 200 Video Shot Insertions as a Function of Minimum Utilization

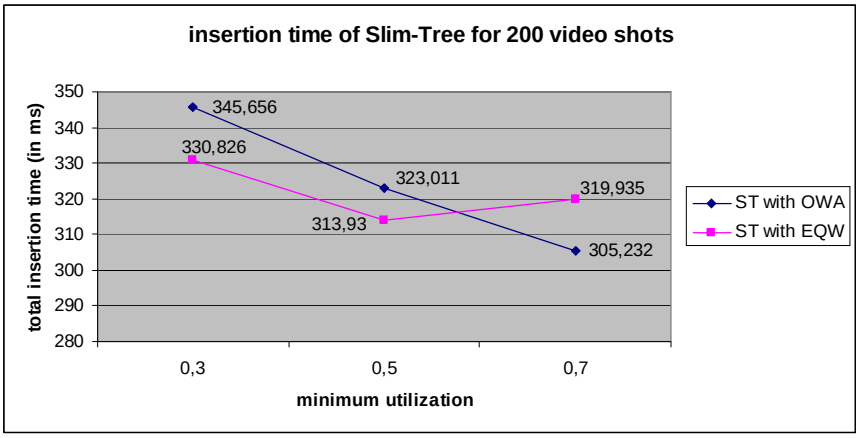

Figure 7.47 Total Insertion Time of Slim-Tree for 200 Video Shots as a Function of Minimum Utilization

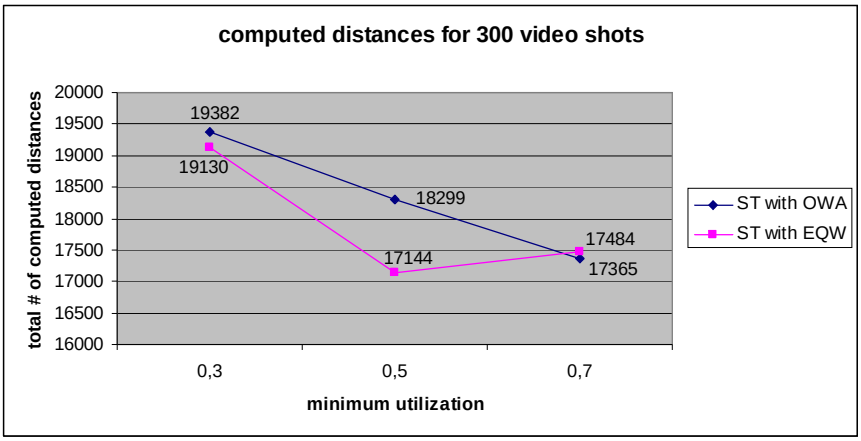

Figure 7.48 # of Computed Distances for 300 Video Shot Insertions as a Function of Minimum Utilization

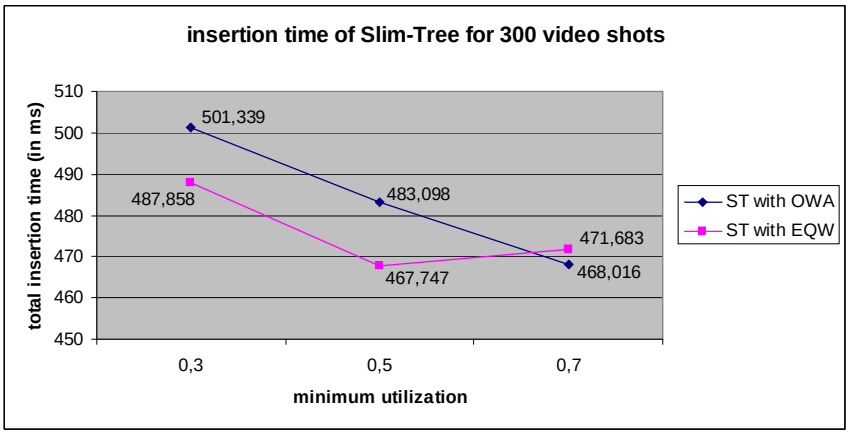

Figure 7.49 Total Insertion Time of Slim-Tree for 300 Video Shots as a Function of Minimum Utilization

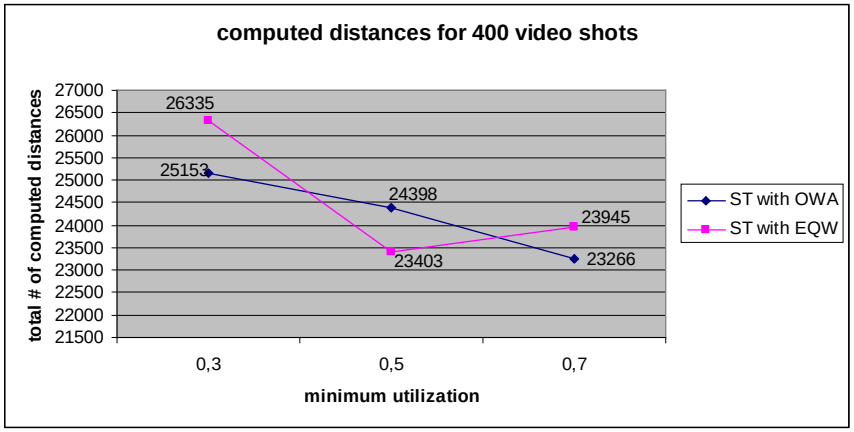

Figure 7.50 # of Computed Distances for 400 Video Shot Insertions as a Function of Minimum Utilization

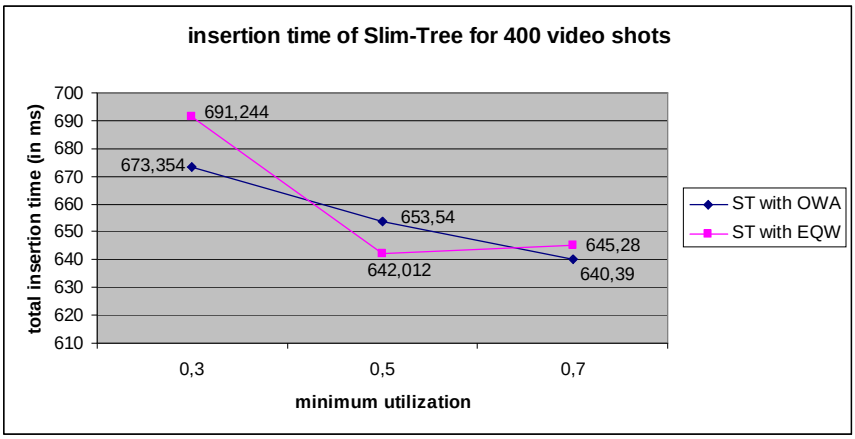

Figure 7.51 Total Insertion Time of Slim-Tree for 400 Video Shots as a Function of Minimum Utilization

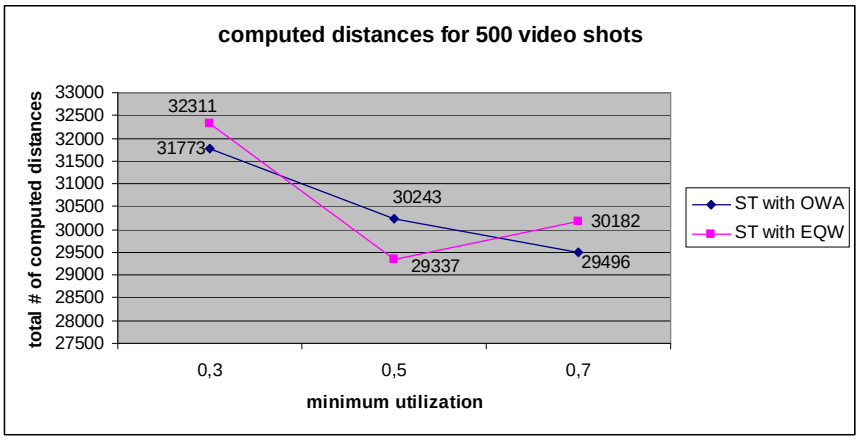

Figure 7.52 # of Computed Distances for 500 Video Shot Insertions as a Function of Minimum Utilization

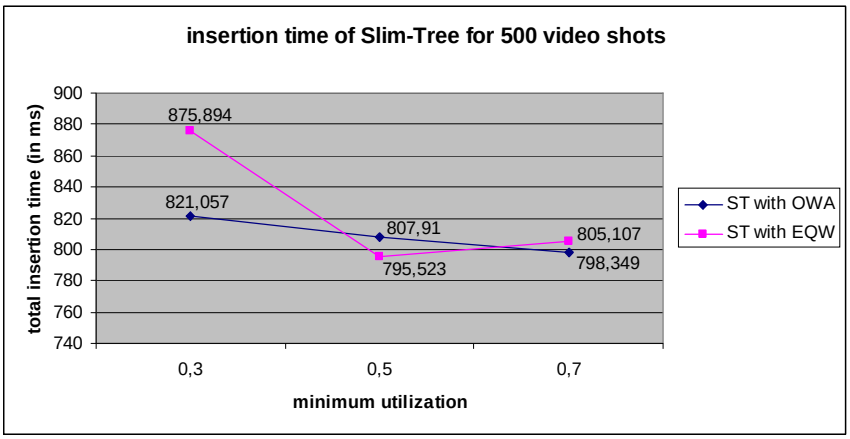

Figure 7.53 Total Insertion Time of Slim-Tree for 500 Video Shots as a Function of Minimum Utilization

As seen from the insertion test results of index structures (from Figure 7.43 to Figure 7.53), insertion performance of BitMatrix for video shots is completely parallel to the performance of BitMatrix for images. BitMatrix performs better than Slim-Tree using OWA or EQW. The BitMatrix insertion time increases as the number of processed objects is increased and using OWA or EQW does not affect the insertion times as expected. Slim-Tree with OWA performs better than Slim-Tree with EQW in terms of number of distance computations and elapsed time with high minimum utilization values such as 0.7. This result for Slim-Tree is different from the results of images, since for video shots in addition to color, texture and shape features, motion feature is indexed by Slim-Tree and different weights are used in OWA operator.

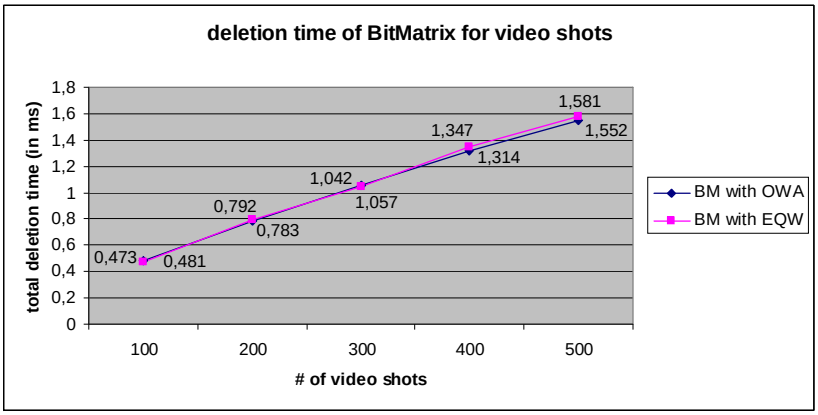

Figure 7.54 Total Deletion Time of BitMatrix for Video Shots as a Function of # of Video Shots

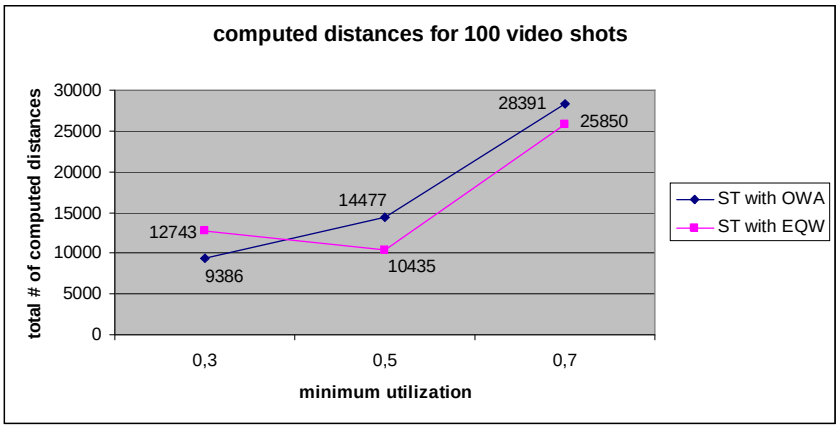

Figure 7.55 # of Computed Distances for 100 Video Shot Deletions as a Function of Minimum Utilization

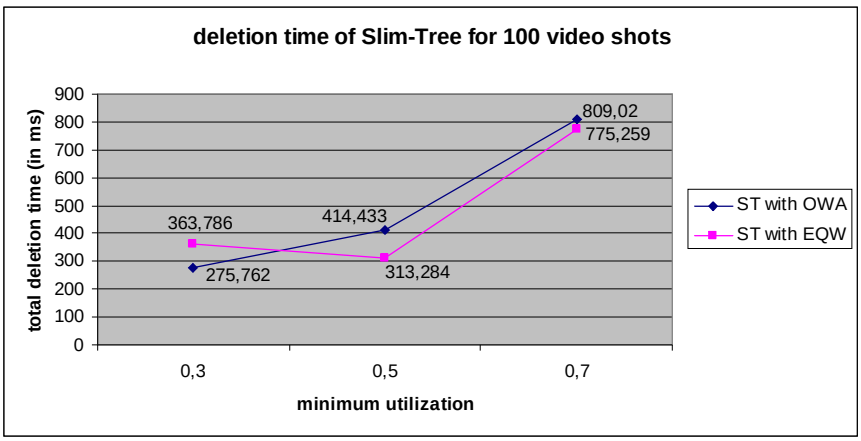

Figure 7.56 Total Deletion Time of Slim-Tree for 100 Video Shots as a Function of Minimum Utilization

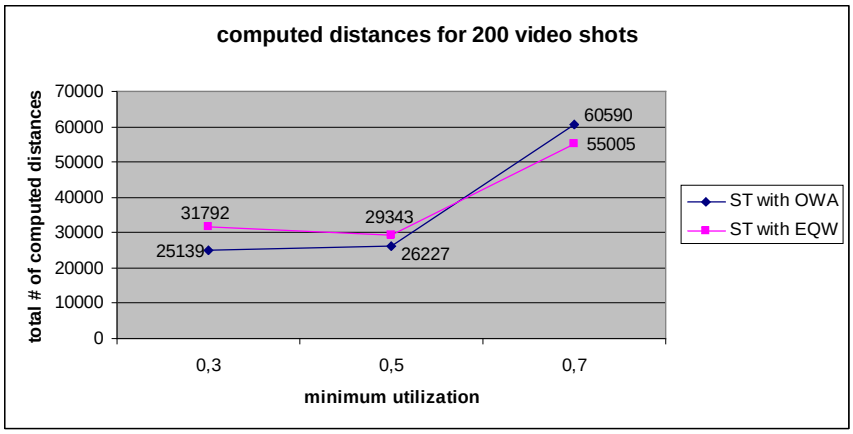

Figure 7.57 # of Computed Distances for 200 Video Shot Deletions as a Function of Minimum Utilization

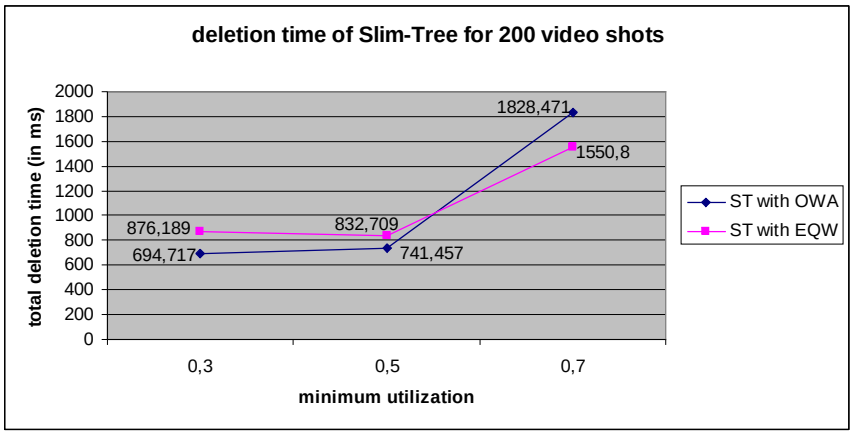

Figure 7.58 Total Deletion Time of Slim-Tree for 200 Video Shots as a Function of Minimum Utilization

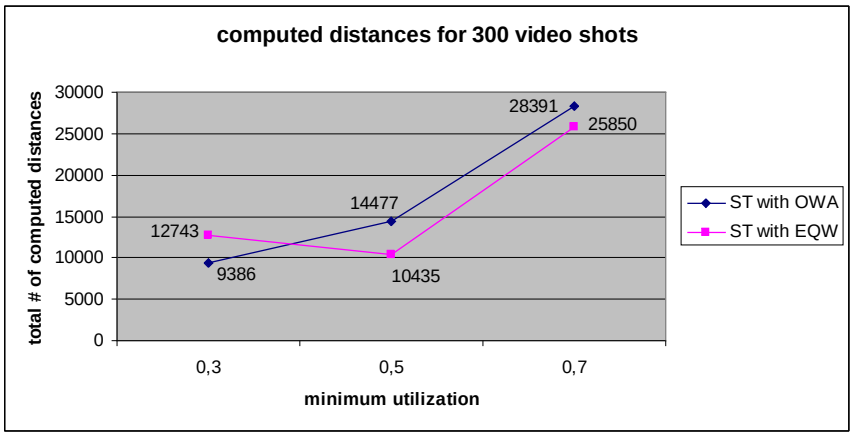

Figure 7.59 # of Computed Distances for 300 Video Shot Deletions as a Function of Minimum Utilization

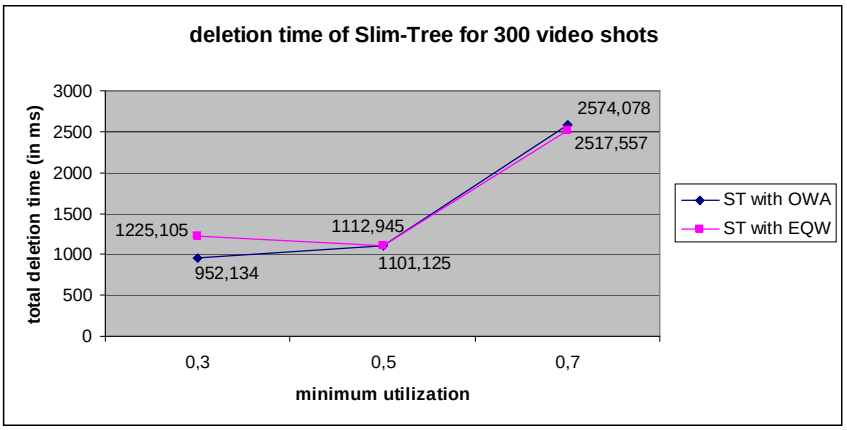

Figure 7.60 Total Deletion Time of Slim-Tree for 300 Video Shots as a Function of Minimum Utilization

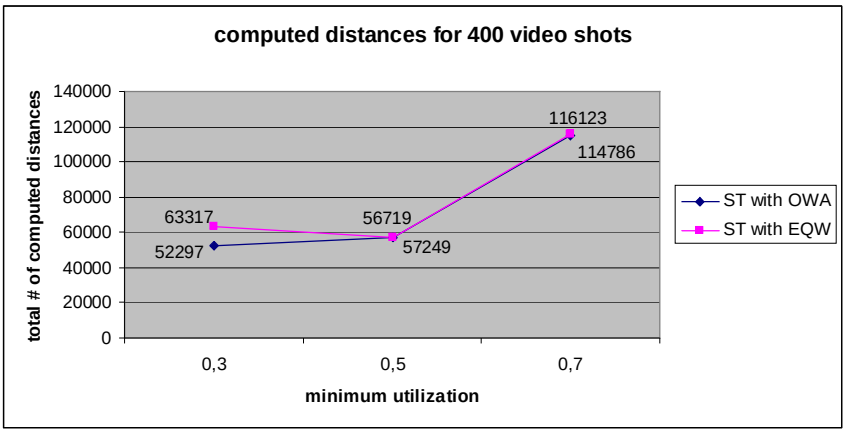

Figure 7.61 # of Computed Distances for 400 Video Shot Deletions as a Function of Minimum Utilization

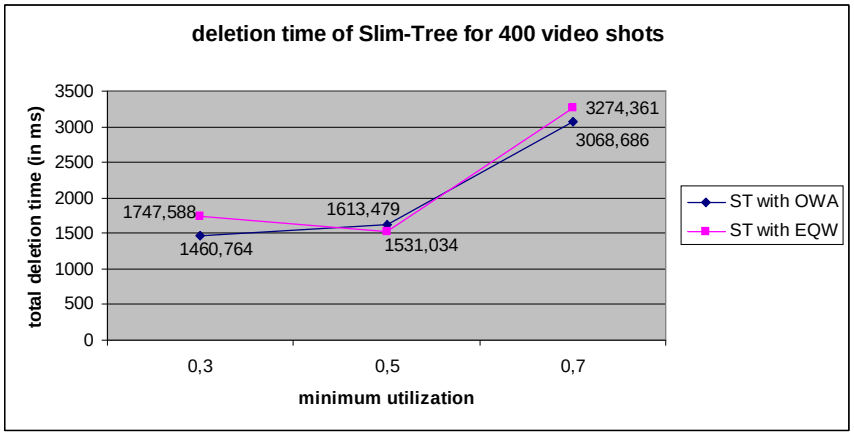

Figure 7.62 Total Deletion Time of Slim-Tree for 400 Video Shots as a Function of Minimum Utilization

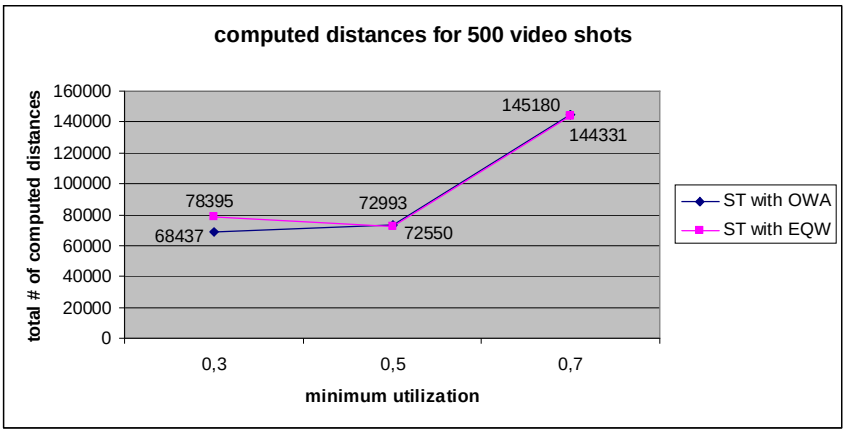

Figure 7.63 # of Computed Distances for 500 Video Shot Deletions as a Function of Minimum Utilization

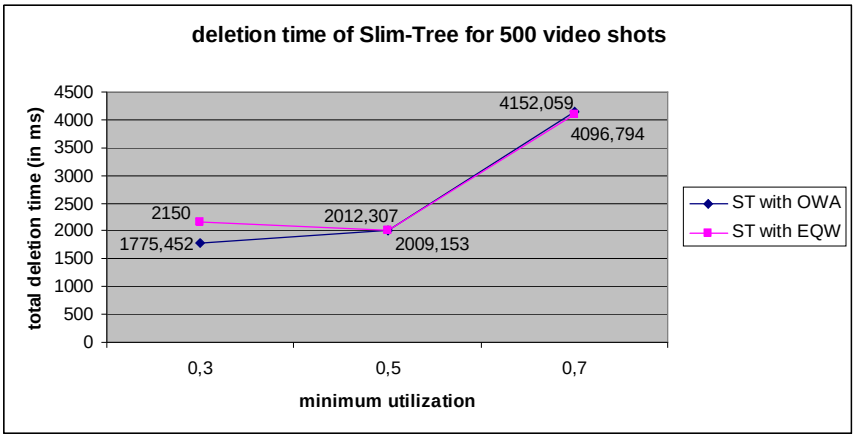

Figure 7.64 Total Deletion Time of Slim-Tree for 500 Video Shots as a Function of Minimum Utilization

As seen from the deletion test results of index structures (from Figure 7.54 to Figure 7.64), deletion performance of BitMatrix for video shots is completely parallel to the performance of BitMatrix for images. BitMatrix performs much better than Slim-Tree using OWA or EQW. Slim-Tree with OWA performs better than Slim-Tree with EQW in terms of number of distance computations and elapsed time for deletion with low minimum utilization values such as 0.3 contrary to insertion test results of Slim-Tree for video shots. This result for Slim-Tree is also different from the results of images, since for video shots in addition to color, texture and shape features, motion feature is indexed by Slim-Tree and different weights are used in OWA operator.

7.3 Querying the Index Structures

7.3.1 Retrieval Efficiency

The retrieval efficiency of the system was evaluated by using Average Normalized Modified Retrieval Rank (ANMRR) metric [4] and precision/recall values.

7.3.1.1 ANMRR

ANMRR can be defined as follows [4]:

$NG(q)$: the number of the ground truth objects (expected result objects) for a query q.

$K(q)$ = min(4 * NG(q), 2 * max{NG(q)} of all q's)

$Rank(k)$: the rank of an object k in retrieval results and is defined as:

$$\text{Rank(k)} = \begin{cases} \text{Rank (k)}, & \text{if Rank(k)} \le \text{K(q)} \\ 1.25 * \text{K(q)}, & \text{if Rank(k)} > \text{K(q)} \end{cases}$$

Using the definition of Rank (k), *Average Rank (AVR)* for query q is defined as:

$$AVR\ (q) = \frac{1}{NG\ (q)} \sum_{k=1}^{NG\ (q)} Rank\ (k) \qquad \textbf{(18)}$$

AVR(q) depends on the size of the ground truth, NG(q). To reduce the influences of NG(q), Modified Retrieval Rank (MRR) is defined that is as follows:

$$MRR\ (q) = AVR\ (q) - 0.5 * \left[1 + NG\ (q)\right] \qquad \textbf{(19)}$$

MRR(q) is always greater than or equal to zero. However, MRR(q)'s upper bound is still dependent on NG(q). This leads to definition of *Normalized Modified Retrieval Rank (NMRR)* that is defined as:

$$NMRR\ (q) = \frac{MRR\ (q)}{1.25\ *\ K\ (q) - 0.5\ *\ \left[1 + NG\ (q)\right]} \qquad \textbf{(20)}$$

NMRR(q) is in range [0-1], 0 indicates whole ground truth found where 1 indicates no object in the ground truth found. By using NMRR values of all *q*'s, we get the *Average Normalized Modified Retrieval Rank (ANMRR)* that is used as the evaluation criteria for MPEG-7 visual descriptors. ANMRR is defined as:

$$ANMRR = \frac{1}{NQ}\sum_{q=1}^{NQ} NMRR(q) \quad \textbf{(21)}$$

where NQ is the number of queries.

7.3.1.2 Precision and Recall

In this study, precision is computed as the fraction of the retrieved objects that are relevant (i.e. in the ground truth of the query object) and can be defined as:

$$P = \frac{|relevant \cap retrieved|}{|retrieved|} \quad \textbf{(22)}$$

In this study, recall is computed as the fraction of the relevant (i.e. in the ground truth of the query object) objects that are successfully retrieved and can be defined as:

$$R = \frac{|relevant \cap retrieved|}{|relevant|} \quad \textbf{(23)}$$

7.3.1.3 Results for Images

We performed 100 queries over an image database of 1000 Corel images for the two types (i.e. using EQW and OWA approaches for similarity measurement) of BitMatrix and Slim-Tree and computed the ANMRR values of BitMatrix and Slim-Tree using OWA/EQW. We also computed the ANMRR values for Sequential Scan in order to compare with the ANMRR values of BitMatrix and Slim-Tree. The results are shown in Table 7.1.

Table 7.1 ANMRR Results for 100 Image Queries over 1000 Images

	OWA	**EQW**
BitMatrix	0.245247	0.254097
SlimTree	0.26793	0.27053
Sequential Scan	0.2802	0.28131

As seen from Table 7.1, BitMatrix using OWA has the best ANMRR values, Slim-Tree using OWA has better ANMRR values than Slim-Tree using EQW and finally both BitMatrix and Slim-Tree using OWA/EQW have better ANMRR values than Sequential Scan using OWA/EQW. In addition to ANMRR, precision/recall values of the two types (i.e. using EQW and OWA approaches for similarity measurement) of BitMatrix and Slim-Tree were computed. As in ANMRR, precision/recall values for Sequential Scan were computed in order to compare with the precision/recall values of BitMatrix and Slim-Tree. In precision/recall tests, we retrieved top twenty objects that are similar to the query object. The results are shown in Table 7.2.

Table 7.2 Precision and Recall Values for 100 Image Queries over 1000 Images

	Precision		**Recall**	
	OWA	**EQW**	**OWA**	**EQW**
BitMatrix	0.328	0.3182	0.8062	0.784
SlimTree	0,3191	0,3178	0.7916	0.776
Seq. Scan	0,3183	0,3175	0.7889	0.7748

As seen from Table 7.2, BitMatrix using OWA has the best precision/recall values, Slim-Tree using OWA has better precision/recall values than Slim-Tree using EQW and finally both BitMatrix and Slim-Tree using OWA/EQW have better precision/recall values than Sequential Scan using OWA/EQW.

7.3.1.4 Results for Videos

We performed 50, 100, 150, 200, 250 and 300 queries over a video database of 1000 video shots for the two types (i.e. using EQW and OWA approaches for similarity measurement) of BitMatrix and Slim-Tree and computed the ANMRR values of BitMatrix and Slim-Tree using OWA/EQW. We also computed the ANMRR values for Sequential Scan in order to compare with the ANMRR values of BitMatrix and Slim-Tree. Figure 7.65 shows the effects of using OWA on the ANMRR values of BitMatrix and Table 7.3 gives the ANMRR values of queries for BitMatrix.

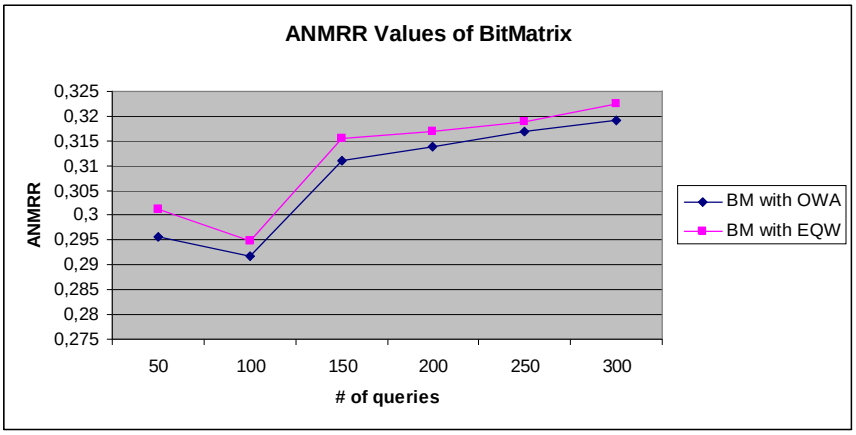

Figure 7.65 ANMRR Values of BitMatrix Using OWA and EQW

Table 7.3 ANMRR Values of BitMatrix – Naïve Approach (ct = 2)

	BitMatrix with OWA	BitMatrix with EQW
50	0.295754	0.301393
100	0.291802	0.294951
150	0.31097	0.315452
200	0.313773	0.316939
250	0.316788	0.318805
300	0.319192	0.322388

As seen from Figure 7.65 and Table 7.3, BitMatrix using OWA with naïve approach has better ANMRR values than BitMatrix using EQW with naïve approach. We also performed tests to compare the ANMRR values of naïve and range expansion approaches of BitMatrix. The comparison of ANMRR values of BitMatrix using OWA/EQW with naïve and range expansion approach is given in Figure 7.66. The ANMRR values of BitMatrix with range expansion approach are given in Table 7.4 (cardinality threshold = 2, expansion threshold = 0.1).

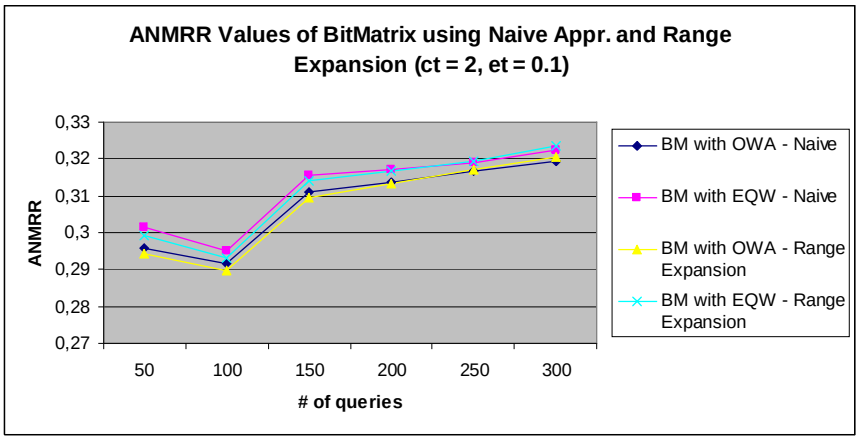

Figure 7.66 ANMRR Values of BitMatrix Using OWA – Naïve Approach and Range Expansion (ct = 2)

Table 7.4 ANMRR Values of BitMatrix – Range Expansion (ct = 2, et = 0.1)

	BitMatrix with OWA	BitMatrix with EQW
50	0.294215	0.299403
100	0.289775	0.293164
150	0.309366	0.314013
200	0.313198	0.316568
250	0.31705	0.319197
300	0.320372	0.323361

As seen from Figure 7.66, Table 7.3 and Table 7.4, when cardinality threshold is 2, BitMatrix using OWA/EQW with naïve approach has similar ANMRR values with BitMatrix using OWA/EQW with range expansion approach. Thus, we performed further tests on ANMRR values of BitMatrix by increasing the cardinality threshold (i.e. by setting cardinality threshold to 3) to see the effects of using range expansion on BitMatrix. The ANMRR values of BitMatrix using OWA/EQW with naïve and range expansion approach are given in Table 7.5 and Table 7.6, respectively. The comparison of ANMRR values of BitMatrix using OWA/EQW with naïve and range expansion approach is given in Figure 7.67.

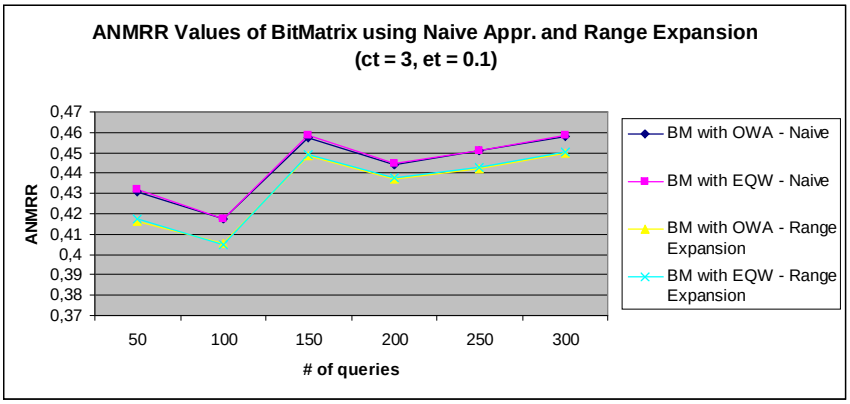

Figure 7.67 ANMRR Values of BitMatrix Using OWA – Naïve Approach and Range Expansion (ct = 3)

Table 7.5 ANMRR Values of BitMatrix – Naïve Approach (ct = 3)

	BitMatrix with OWA	BitMatrix with EQW
50	0.430675	0.431778
100	0.417294	0.417188
150	0.457496	0.458333
200	0.443897	0.444409
250	0.45081	0.45122
300	0.458243	0.458704

Table 7.6 ANMRR Values of BitMatrix – Range Expansion (ct = 3, et = 0.1)

	BitMatrix with OWA	BitMatrix with EQW
50	0.416481	0.417411
100	0.405159	0.404812
150	0.448641	0.449302
200	0.437284	0.437693
250	0.442351	0.442614
300	0.449962	0.450245

As seen from Figure 7.67, Table 7.5 and Table 7.6, when cardinality threshold is 3, BitMatrix using OWA/EQW with range expansion approach has better ANMRR values than BitMatrix using OWA/EQW with naïve approach. It is also observed that when cardinality threshold is increased, the difference between the ANMRR values of BitMatrix using OWA and EQW becomes smaller. In addition, when the cardinality threshold is increased, the ANMRR values of BitMatrix also increase. Since, BitMatrix eliminates the related video shots with the unrelated ones because of the high cardinality threshold value.

Figure 7.68 shows the effects of using OWA on the ANMRR values of Slim-Tree and Table 7.7 gives the ANMRR values of queries for Slim-Tree. As seen from Figure 7.68 and Table 7.7, Slim-Tree using OWA has better ANMRR values than Slim-Tree using EQW.

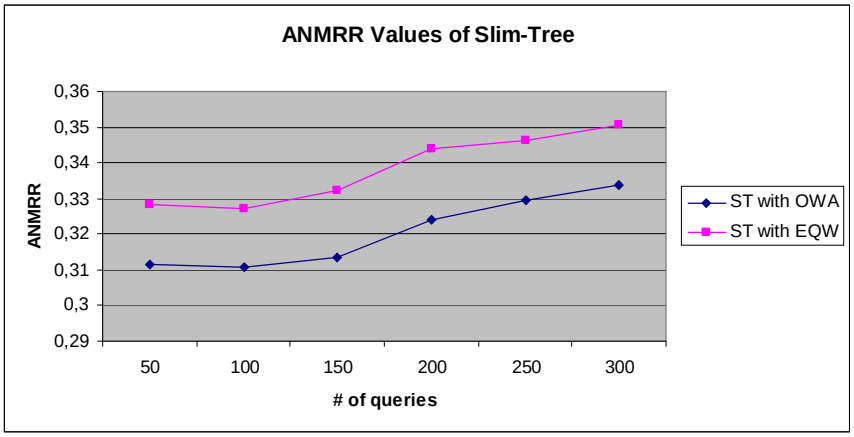

Figure 7.68 ANMRR Values of Slim-Tree Using OWA and EQW

Table 7.7 ANMRR Values of Slim-Tree

	Slim-Tree with OWA	Slim-Tree with EQW
50	0.311452	0.328292
100	0.310823	0.326989
150	0.31365	0.332285
200	0.324042	0.343846
250	0.329454	0.346372
300	0.333784	0.350626

Figure 7.69 shows the effects of using OWA on the ANMRR values of Sequential Scan and Table 7.8 gives the ANMRR values of queries for Sequential Scan. As seen from Figure 7.68 and Table 7.8, parallel to the results of BitMatrix and Slim-Tree, Sequential Scan using OWA has better ANMRR values than Sequential Scan using EQW.

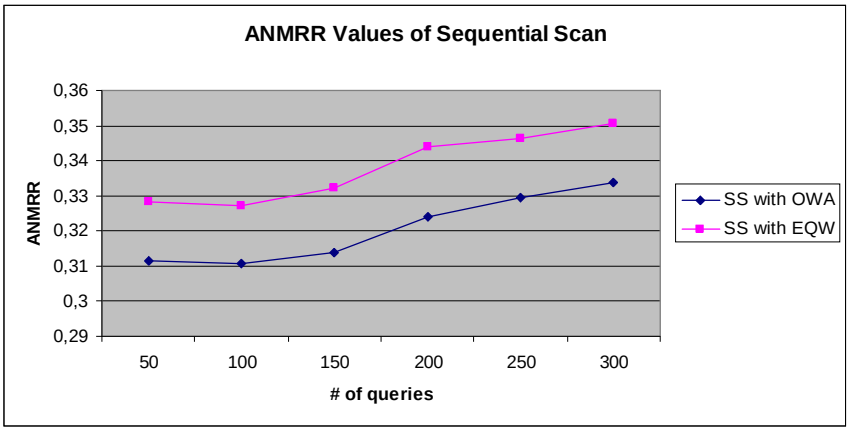

Figure 7.69 ANMRR Values of Sequential Scan Using OWA and EQW

Table 7.8 ANMRR Values of Sequential Scan

	Seq. Scan with OWA	Seq. Scan with EQW
50	0.311456	0.328292
100	0.310825	0.326989
150	0.313664	0.332285
200	0.324053	0.343846
250	0.329465	0.346372
300	0.333806	0.350626

Figure 7.70 shows the comparison of the ANMRR values of BitMatrix, Slim-Tree and Sequential Scan using OWA.

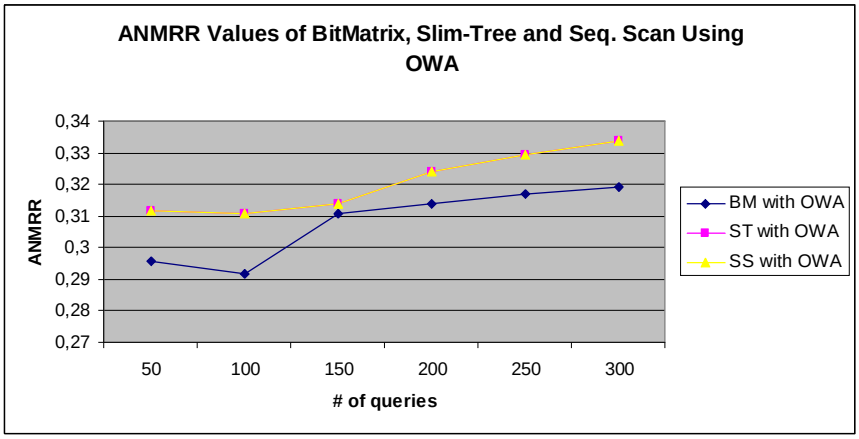

Figure 7.70 Comparison of ANMRR Values of BitMatrix, Slim-Tree and Seq. Scan Using OWA

As seen from Figure 7.70, BitMatrix using OWA has the best ANMRR values when compared to Slim-Tree and Sequential Scan using OWA. It is also observed that Slim-Tree and Sequential Scan using OWA have similar ANMRR values.

Figure 7.71 shows the comparison of the ANMRR values of BitMatrix, Slim-Tree and Sequential Scan using EQW.

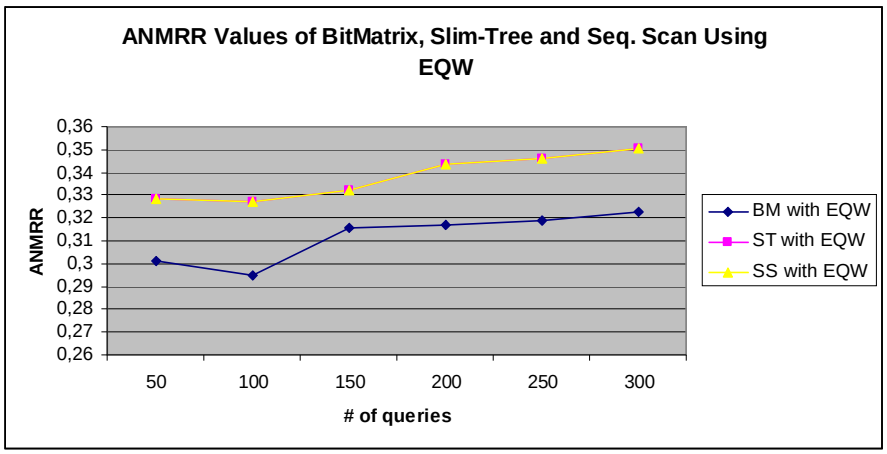

Figure 7.71 Comparison of ANMRR Values of BitMatrix, Slim-Tree and Seq. Scan Using EQW

As seen from Figure 7.71, parallel to the comparison results of ANMRR values in Figure 7.70 , BitMatrix using EQW has the best ANMRR values when compared to Slim-Tree and Sequential Scan using EQW. It is also observed that Slim-Tree and Sequential Scan using EQW have similar ANMRR values.

In addition to ANMRR, precision/recall values of the two types (i.e. using EQW and OWA approaches for similarity measurement) of BitMatrix and Slim-Tree were computed. As in ANMRR, precision/recall values for Sequential Scan were computed in order to compare with the precision/recall values of BitMatrix and Slim-Tree. In precision/recall tests, we retrieved top fifty objects that are similar to the query object.

Figure 7.72 and Figure 7.73 show the effects of using OWA on the precision/recall values of BitMatrix and Table 7.9 gives the precision/recall values of queries for BitMatrix.

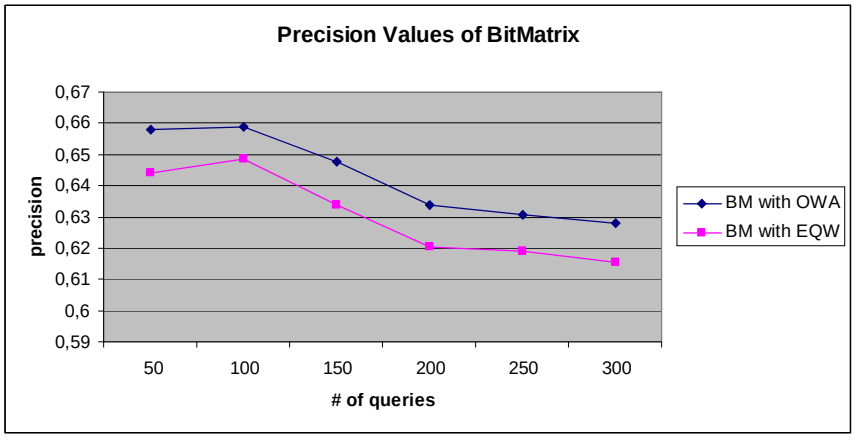

Figure 7.72 Precision Values of BitMatrix Using OWA and EQW

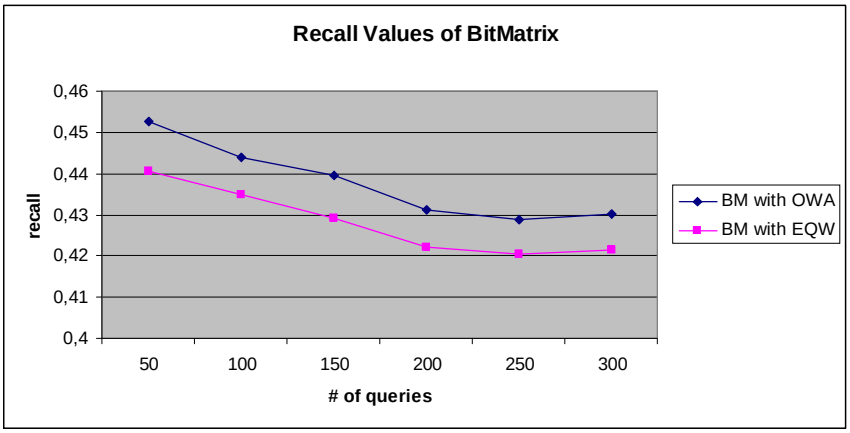

Figure 7.73 Recall Values of BitMatrix Using OWA and EQW

Table 7.9 Precision and Recall Values of BitMatrix – Naïve Approach (ct = 2)

	BitMatrix with OWA		BitMatrix with EQW	
	Precision	**Recall**	**Precision**	**Recall**
50	0.658	0.452473	0.644	0.440539
100	0.659	0.443858	0.6486	0.434844
150	0.6476	0.439659	0.634	0.429024
200	0.6339	0.43134	0.6205	0.422008
250	0.63072	0.428816	0.61904	0.420355
300	0.627867	0.430305	0.6154	0.421297

As seen from Figure 7.72, Figure 7.73 and Table 7.9, BitMatrix using OWA with naïve approach has better precision/recall values than BitMatrix using EQW with naïve approach. We also performed tests to compare the precision/recall values of naïve and range expansion approaches of BitMatrix. The comparisons of precision/recall values of

BitMatrix using OWA/EQW with naïve and range expansion approach are given in Figure 7.74 and Figure 7.75, respectively. The precision/recall values of BitMatrix with range expansion approach are given in Table 7.10 (cardinality threshold = 2, expansion threshold = 0.1).

Table 7.10 Precision and Recall Values of BitMatrix – Range Expansion (ct = 2, et = 0.1)

	BitMatrix with OWA		BitMatrix with EQW	
	Precision	**Recall**	**Precision**	**Recall**
50	0.6596	0.453395	0.6456	0.441409
100	0.6594	0.445437	0.6488	0.435814
150	0.647733	0.440926	0.634	0.429865
200	0.6336	0.431873	0.6205	0.422638
250	0.63048	0.429232	0.61864	0.42061
300	0.628	0.430807	0.614867	0.421339

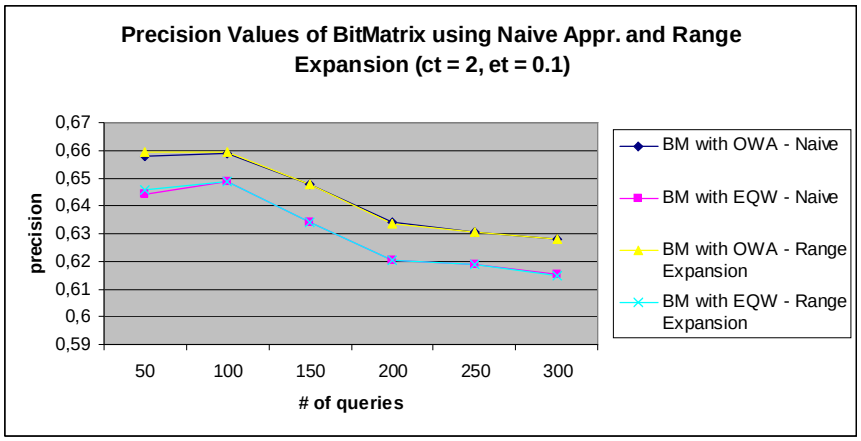

Figure 7.74 Precision Values of BitMatrix Using OWA – Naïve Approach and Range Expansion (ct = 2)

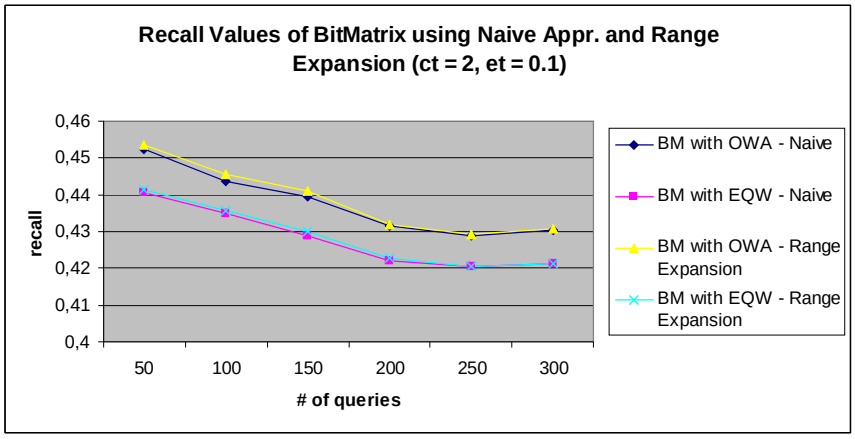

Figure 7.75 Recall Values of BitMatrix Using OWA – Naïve Approach and Range Expansion (ct = 2)

As seen from Table 7.9, Table 7.10, Figure 7.74 and Figure 7.75, when cardinality threshold is 2, BitMatrix using OWA/EQW with naïve approach has similar precision/recall values with BitMatrix using OWA/EQW with range expansion approach. Thus, we performed further tests on precision/recall values of BitMatrix by increasing the cardinality threshold (i.e. by setting cardinality threshold to 3) to see the effects of using range expansion on BitMatrix. The precision/recall values of BitMatrix using OWA/EQW with naïve and range expansion approach are given in Table 7.11 and Table 7.12, respectively. The comparisons of precision/recall values of BitMatrix using OWA/EQW with naïve and range expansion approach are given in Figure 7.76 and Figure 7.77.

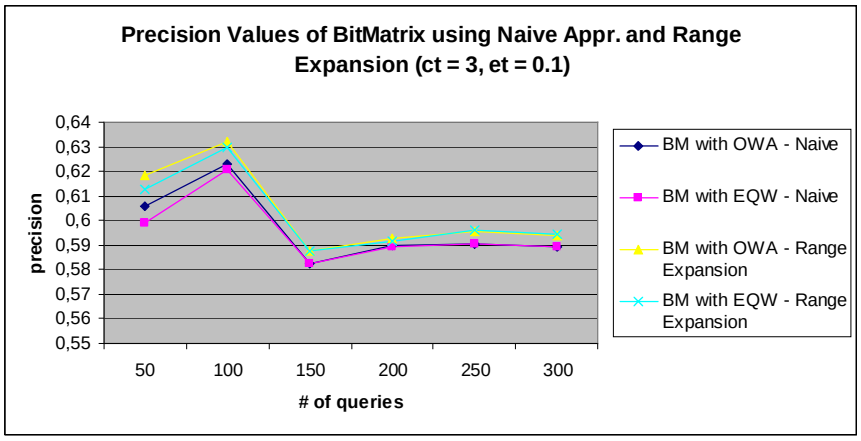

Figure 7.76 Precision Values of BitMatrix Using OWA – Naïve Approach and Range Expansion (ct = 3)

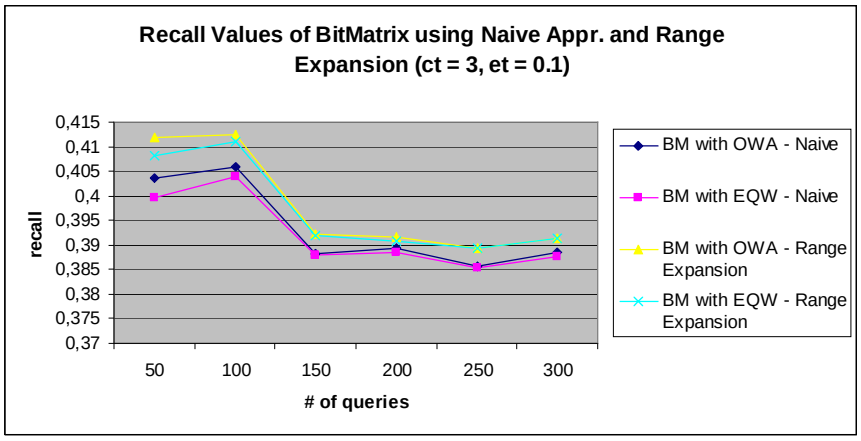

Figure 7.77 Recall Values of BitMatrix Using OWA – Naïve Approach and Range Expansion (ct = 3)

Table 7.11 Precision and Recall Values of BitMatrix – Naïve Approach (ct = 3)

	BitMatrix with OWA		BitMatrix with EQW	
	Precision	**Recall**	**Precision**	**Recall**
50	0.6056	0.403722	0.5992	0.399537
100	0.6228	0.405767	0.6204	0.404031
150	0.5824	0.388315	0.582533	0.387853
200	0.5897	0.389327	0.5891	0.388539
250	0.5904	0.385684	0.59056	0.385409
300	0.589533	0.388373	0.589133	0.387773

Table 7.12 Precision and Recall Values of BitMatrix – Range Expansion (ct = 3, et = 0.1)

	BitMatrix with OWA		BitMatrix with EQW	
	Precision	**Recall**	**Precision**	**Recall**
50	0.6184	0.411789	0.6128	0.408039
100	0.6318	0.412475	0.6298	0.410929
150	0.587733	0.392131	0.587733	0.391828
200	0.5927	0.391673	0.5918	0.390918
250	0.59584	0.389404	0.596	0.389306
300	0.593867	0.391406	0.5942	0.391441

As seen from Figure 7.76 and Figure 7.77, when cardinality threshold is 3, BitMatrix using OWA/EQW with range expansion approach has better precision/recall values than BitMatrix using OWA/EQW with naïve approach. It is also observed that when cardinality threshold is increased, the difference between precision/recall values of BitMatrix using OWA and EQW becomes smaller. In addition, when the cardinality threshold is increased,

the precision/recall values of BitMatrix decrease. Since, BitMatrix eliminates the related video shots with the unrelated ones because of the high cardinality threshold value.

Figure 7.78 and Figure 7.79 show the effects of using OWA on the precision/recall values of Slim-Tree and Table 7.13 gives the precision/recall values of queries for Slim-Tree. As seen from Figure 7.78, Figure 7.79 and Table 7.13, Slim-Tree using OWA has better precision/recall values than Slim-Tree using EQW.

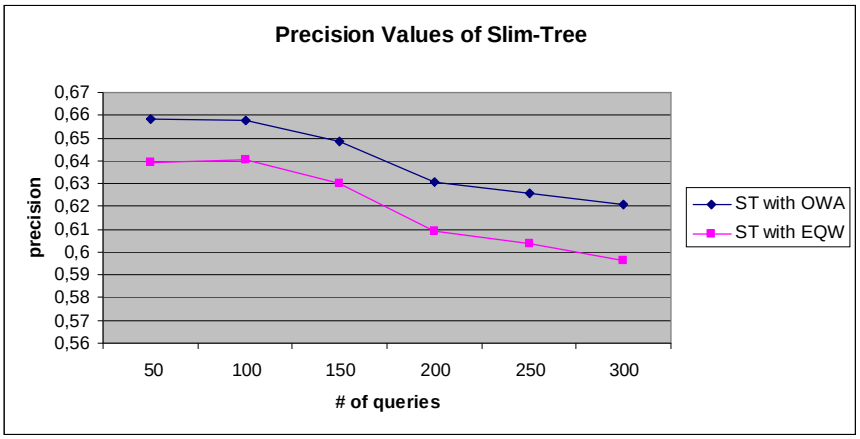

Figure 7.78 Precision Values of Slim-Tree Using OWA and EQW

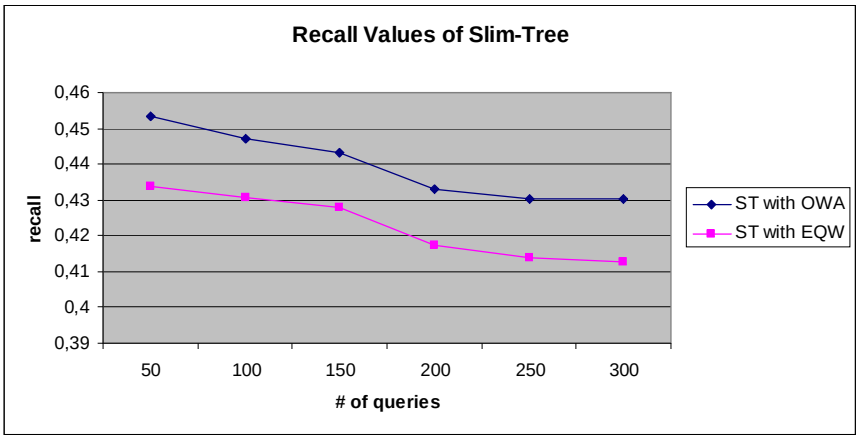

Figure 7.79 Recall Values of Slim-Tree Using OWA and EQW

Table 7.13 Precision and Recall Values of Slim-Tree

	Slim-Tree with OWA		**Slim-Tree with EQW**	
	Precision	**Recall**	**Precision**	**Recall**
50	0.6584	0.453267	0.6392	0.433943
100	0.658	0.447122	0.6408	0.430566
150	0.648533	0.443234	0.630267	0.42781

	Slim-Tree with OWA		Slim-Tree with EQW	
	Precision	**Recall**	**Precision**	**Recall**
200	0.6308	0.433044	0.609	0.417407
250	0.62576	0.430396	0.60352	0.414003
300	0.620533	0.430418	0.596533	0.412515

Figure 7.80 and Figure 7.81 show the effects of using OWA on the precision/recall values of Sequential Scan and Table 7.14 gives the precision/recall values of queries for Sequential Scan. As seen from Figure 7.80, Figure 7.81 and Table 7.14, parallel to the results of BitMatrix and Slim-Tree, Sequential Scan using OWA has better precision/recall values than Sequential Scan using EQW.

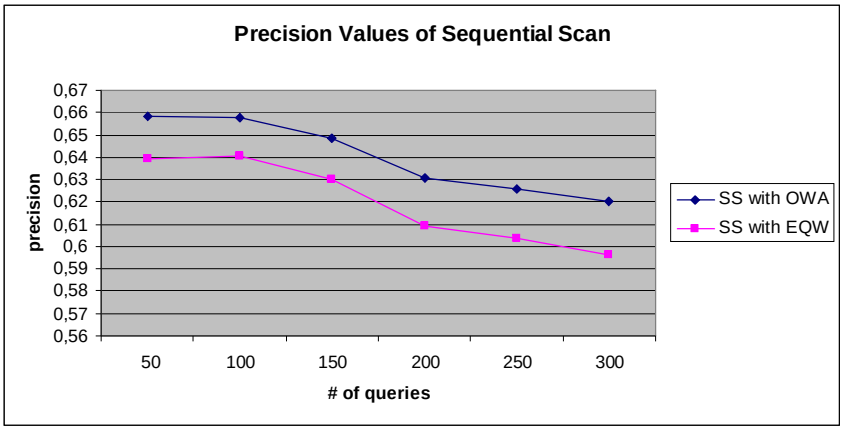

Figure 7.80 Precision Values of Sequential Scan Using OWA and EQW

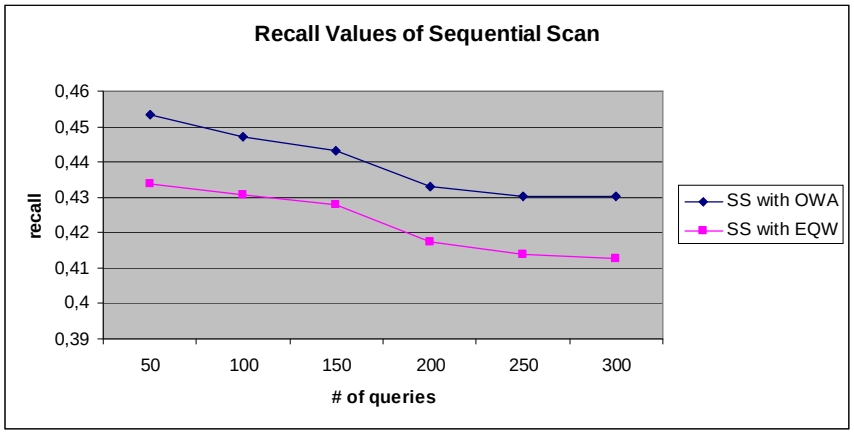

Figure 7.81 Recall Values of Sequential Scan Using OWA and EQW

Table 7.14 Precision and Recall Values of Sequential Scan

	Seq. Scan with OWA		Seq. Scan with EQW	
	Precision	**Recall**	**Precision**	**Recall**
50	0.6584	0.453267	0.6392	0.433943
100	0.658	0.447122	0.6408	0.430566
150	0.6484	0.443162	0.630267	0.42781
200	0.6307	0.43299	0.609	0.417407
250	0.62568	0.430353	0.60352	0.414003
300	0.620333	0.430292	0.596533	0.412515

Figure 7.82 and Figure 7.83 show the comparison of the precision/recall values of BitMatrix, Slim-Tree and Sequential Scan using OWA. As seen from Figure 7.82 and Figure 7.83, BitMatrix, Slim-Tree and Sequential Scan using OWA have similar precision/recall values, except that the precision values of BitMatrix become better for 200, 250 and 300 queries and the recall values of BitMatrix are a bit smaller than Slim-Tree and Sequential Scan.

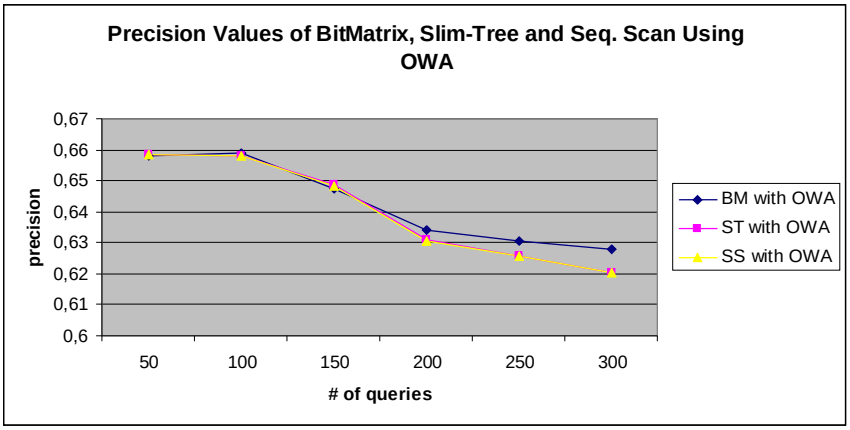

Figure 7.82 Comparison of Precision Values of BitMatrix, Slim-Tree and Seq. Scan Using OWA

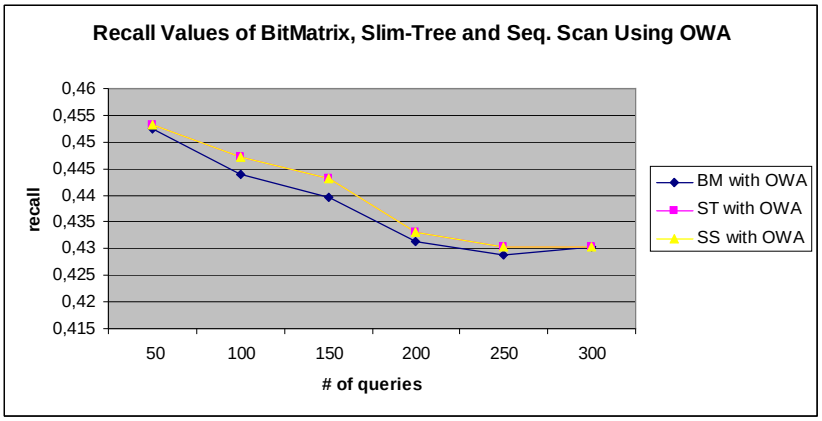

Figure 7.83 Comparison of Recall Values of BitMatrix, Slim-Tree and Seq. Scan Using OWA

Figure 7.84 and Figure 7.85 show the comparison of the precision/recall values of BitMatrix, Slim-Tree and Sequential Scan using EQW. As seen from Figure 7.84 and Figure 7.85, BitMatrix using EQW has better precision/recall values when compared to Slim-Tree and Sequential Scan using EQW. It is also observed that Slim-Tree and Sequential Scan using EQW have similar precision/recall values.

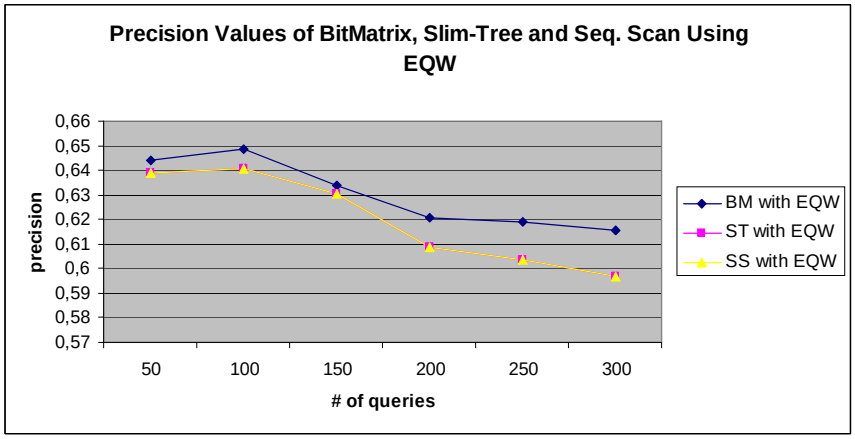

Figure 7.84 Comparison of Precision Values of BitMatrix, Slim-Tree and Seq. Scan Using EQW

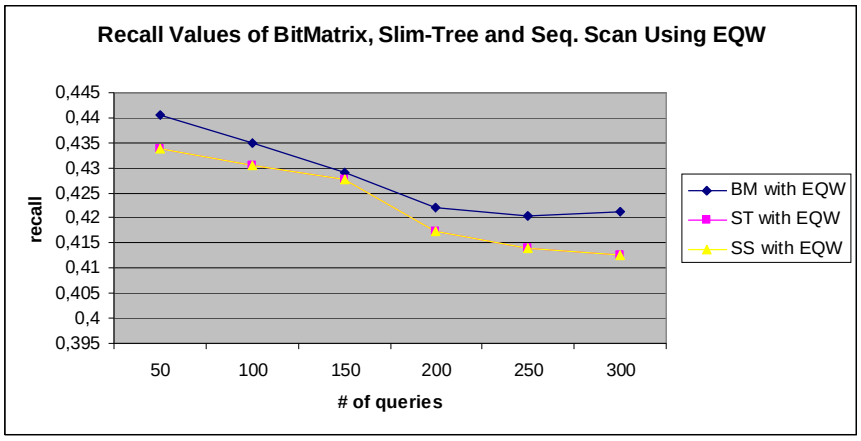

Figure 7.85 Comparison of Recall Values of BitMatrix, Slim-Tree and Seq. Scan Using EQW

7.3.2 k-NN Query

To evaluate the efficiency of k-NN queries by using BitMatrix, Slim-Tree and Sequential Scan, firstly we performed 1000 queries to retrieve the top ten images (10-NN query) over an image database of 1000 images. Table 7.15 reports the retrieval times and number of distance computations of each structure for k-Nearest Neighbor (k-NN) queries.

Table 7.15 Query Response Time and # of Distance Computations for 10-NN Queries over 1000 Images

Index Structure	Storage utilization	# of Dist. Comp.		Resp. Time (ms)	
		min	max	min	Max
BitMatrix with OWA	N/A	71	366	2.764	15.74
BitMatrix with EQW	N/A	71	366	2.833	15.588
SlimTree with OWA	0.7	499	997	14.853	39.709
SlimTree with EQW	0.7	578	1000	15.363	40.009
Seq. Scan with OWA	N/A	1000	1000	26.4	43.692
Seq. Scan with EQW	N/A	1000	1000	26.459	43.064

In addition, to evaluate the efficiency of k-NN queries by using BitMatrix, Slim-Tree and Sequential Scan, we also performed 1000 queries to retrieve the top ten video shots (10-NN query) over an video database of 1000 video shots. Table 7.16 reports the retrieval times

and number of distance computations of each structure for k-Nearest Neighbor (k-NN) queries.

Table 7.16 Query Response Time and # of Distance Computations for 10-NN Queries over 1000 Video Shots

Index Structure	Storage utilization	# of Dist. Comp.		Resp. Time (ms)	
		min	max	min	Max
BitMatrix with OWA (Naïve)	N/A	123	438	5.021	19.815
BitMatrix with EQW (Naïve)	N/A	123	438	5.129	20.155
BitMatrix with OWA (Range Expansion)	N/A	123	592	5.013	20.671
BitMatrix with EQW (Range Expansion)	N/A	123	592	4.937	20.585
SlimTree with OWA	0.7	432	985	13.42	46.277
SlimTree with EQW	0.7	646	1000	20.534	45.62
Seq. Scan with OWA	N/A	1000	1000	26.023	45.49
Seq. Scan with EQW	N/A	1000	1000	25.983	44.447

As seen from the results reported in Table 7.15 and Table 7.16, BitMatrix using OWA/EQW makes less distance computations than Slim-Tree and Sequential Scan using OWA/EQW. It is observed from the results that using range expansion on BitMatrix causes more distance computations when compared to naive approach, since range expansion approach eliminates less objects before distance computations. It is also observed from the results that Slim-Tree using OWA makes less distance computations than Slim-Tree using EQW, where BitMatrix using OWA makes same number of distance computations as BitMatrix using EQW. In addition, Slim-Tree and BitMatrix make less distance computations than Sequential Scan.

7.3.3 Range Query

To evaluate the efficiency of range queries by using BitMatrix, Slim-Tree and Sequential Scan, firstly we performed 1000 queries where radius was 0.2 over an image database of 1000 images. Table 7.17 reports the retrieval times and number of distance computations of each structure for range queries where r=0.2.

Table 7.17 Query Response Time and # of Distance Computations for Range Queries over 1000 Images (r = 0.2)

Index Structure	Storage utilization	# of Dist. Comp.		Response Time (ms)	
		min	max	min	max
BitMatrix with OWA	N/A	71	366	2.725	15.554
BitMatrix with EQW	N/A	71	366	2.924	15.501
SlimTree with OWA	0.7	494	1066	15.241	50.606
SlimTree with EQW	0.7	485	1055	14.452	50.689
Seq. Scan with OWA	N/A	1000	1000	25.382	44.413
Seq. Scan with EQW	N/A	1000	1000	25.176	44.23

As seen from the results reported in Table 7.17, BitMatrix using OWA/EQW makes less distance computations than Slim-Tree and Sequential Scan using OWA/EQW. Slim-Tree using OWA and EQW make similar number of distance computations for range queries. In addition, Slim-Tree (in general) and BitMatrix make less distance computations than Sequential Scan. To evaluate the efficiency of range queries by using BitMatrix, Slim-Tree and Sequential Scan, we also performed 1000 queries where radius was 0.2 over a video database of 1000 video shots. Table 7.18 reports the retrieval times and number of distance computations of each structure for range queries where r=0.2.

Table 7.18 Query Response Time and # of Distance Computations for Range Queries over 1000 Video Shots (r = 0.2)

Index Structure	Storage utilization	# of Dist. Comp.		Response Time (ms)	
		min	max	min	max
BitMatrix with OWA (Naïve)	N/A	123	438	5.045	18.017
BitMatrix with EQW (Naïve)	N/A	123	438	4.799	19.619

Index Structure	Storage utilization	# of Dist. Comp.		Response Time (ms)	
		min	max	min	max
BitMatrix with OWA (Range Expansion)	N/A	123	592	4.804	20.811
BitMatrix with EQW (Range Expansion)	N/A	123	592	4.794	20.538
SlimTree with OWA	0.7	519	972	15.146	43.777
SlimTree with EQW	0.7	615	963	17.438	42.474
Seq. Scan with OWA	N/A	1000	1000	25.878	39.941
Seq. Scan with EQW	N/A	1000	1000	25.237	38.326

The results reported in Table 7.18 are parallel to the results of k-NN queries of video shots given in Table 7.16.

7.4 Discussion

In this study, we designed and implemented a content-based video retrieval system that evaluates the similarity of video shots using low-level features (i.e. color, texture, shape and motion). The system includes two different index structures, namely BitMatrix and Slim-Tree for efficient content-based retrieval. Performance tests on building, updating and querying index structures were applied. We performed tests both for image and video retrieval and the test results are summarized in the following paragraphs.

During construction/update of Slim-Tree, the number of distance computations and the elapsed times are the key indicators for the evaluation of the effectiveness of Slim-Tree construction/update. Thus, performance tests on building and updating Slim-Tree contains the number of distance computations and elapsed times for Slim-Tree using OWA and EQW.

During construction/update of BitMatrix, the elapsed times for clustering visual features to generate bitmap signatures is the key indicator for the evaluation of the effectiveness of

BitMatrix construction/update. There is no distance computation during construction /update of BitMatrix, only clustering visual features is performed to generate bitmap signatures. Thus, performance tests on building and updating BitMatrix contains only the elapsed times for Bit-Matrix using OWA and EQW.

We computed the number of distance computations (only for Slim-Tree) and the index construction times of Slim-Tree and BitMatrix for 100, 300, 500, 700 and 1000 video shots and also for 100, 300, 500 and 700 images. Test results for video shots are given in Section 7.1.2 and for images are given in Section 7.1.1.

As seen from the figures in Section 7.1.2, for video shots using OWA on Slim-Tree generally provides an improvement on the number of distance computations and elapsed times for index construction with high minimum utilization values such as 0.7. For example, the number of distance computations for the construction of Slim-Tree using OWA of 1000 video shots with 0.7 as minimum utilization value was 51388 and the index construction time was 1795 ms, where the number of distance computations and index construction time of Slim-Tree using EQW with the same minimum utilization value (0.7) were 51812 and 1875 ms, respectively. As seen from the figures in Section 7.1.1, for images using OWA on Slim-Tree provides an improvement on the number of distance computations and elapsed times for index construction when the number of indexed objects is increased such as 500 and 700 images.

Also, as seen in Figure 7.1 and Figure 7.10, BitMatrix construction time was higher than Slim-Tree using OWA/EQW. The reason for this result was the clustering time of visual features of images/video shots and this clustering time increased as the number of indexed objects increased. Using OWA or EQW on BitMatrix did not affect the index construction times as expected.

We computed the number of distance computations (only for Slim-Tree) and the index update times of Slim-Tree and BitMatrix for 100, 200, 300, 400 and 500 image/video shot insertion/deletion into/from an image/video database of 1000 images/video shots. Test results for video shots are given in Section 7.2.2 and for images are given in Section 7.2.1.

As seen from the figures in Section 7.2.2, for the insertion of video shots as in index construction tests, using OWA on Slim-Tree provides an improvement on the number of distance computations and index update times with high minimum utilization values such as 0.7. For example, during 300 video shot insertions, the number of distance computations for

the Slim-Tree using OWA with 0.7 as minimum utilization value was 17365 and the update time was 468.016 ms, where the number of distance computations and update time of Slim-Tree using EQW with the same minimum utilization value (0.7) were 17484 and 471.683, respectively. For the deletion of video shots, contrary to construction/insertion test results using OWA on Slim-Tree with low minimum utilization values such as 0.3 gave better results than Slim-Tree using EQW in terms of the number of distance computations and update times. For images as seen from the figures in Section 7.2.1, using OWA on Slim-Tree provides an improvement on the number of distance computations and index update times.

As seen in Figure 7.21, Figure 7.32, Figure 7.43 and Figure 7.54, unlike the index construction test results, BitMatrix update time is much lower than Slim-Tree using OWA/EQW. The reasons for insertion test results are doing no distance computation and only generating the bitmap signature of the indexed objects during the insertions into BitMatrix. The reasons for deletion test results are doing no distance computation and only removing the bitmap signature of the deleted objects during the deletions from BitMatrix. As in BitMatrix construction, the index update times increase as the number of inserted/deleted objects increase. Using OWA or EQW on BitMatrix did not affect the update times as expected.

Query performances of Slim-Tree and BitMatrix were evaluated by using k-NN and range queries. Effects of range expansion approach on the query performance of BitMatrix are examined and the results are discussed in Section 7.3, in detail. For Slim-Tree, as in index construction and update, the number of distance computations and elapsed times were used; for BitMatrix the number of distance computations was evaluated in addition to query elapsed time.

We performed 1000 k-NN queries that retrieve the top ten images/video shots that are most similar to the query image/video shot over an image/video database of 1000 images/video shots. The results shown in Table 7.15 and Table 7.16 depict that BitMatrix and Slim-Tree make less distance computations than Sequential Scan and BitMatrix performs better than Slim-Tree in terms of distance computations and query elapsed time. This is because BitMatrix eliminates some images/video shots based on the bitmap signatures of those images/video shots that are generated by using the clustering results of visual features before making distance computations during k-NN queries. The results in Table 7.15 and Table 7.16 also depict that using OWA on Slim-Tree improves the k-NN query performance in terms of distance computations and query elapsed time. Using OWA on

BitMatrix does not affect the k-NN query performance in terms of distance computations and query elapsed time as expected. Because, the number of distance computations and so the query elapsed time are only dependent on the elimination step of images/video shots that is same for both BitMatrix using OWA and EQW.

We performed 1000 range queries that retrieve images/video shots whose distance were within 0.2 to the query image/video shot over an image/video database of 1000 images/video shots. The results shown in Table 7.17 and Table 7.18 depict that BitMatrix performs better than Slim-Tree and Sequential Scan in terms of distance computations and query elapsed time as in k-NN queries. Parallel to k-NN query results, using OWA on Slim-Tree improves the range query performance in terms of distance computations and query elapsed time. Also, Slim-Tree makes less distance computations than Sequential Scan. However, as seen from results in Table 7.17, Slim-Tree can make more distance computations than Sequential Scan for some of the 1000 range queries. A final observation for queries on BitMatrix is the type of query (k-NN or range) does not affect the number of distance computations as expected, since the number of distance computations only depends on the elimination step of images/video shots that is same for both k-NN and range queries on BitMatrix.

For the evaluation of the retrieval efficiency of BitMatrix and Slim-Tree, ANMRR and precision/recall values of BitMatrix and Slim-Tree were computed and compared with Sequential Scan. The results are reported in Section 7.3.1.3 and 7.3.1.4. From the test results, we observed that BitMatrix with OWA adapted has the best results. This is because of the elimination step of images/video shots according to their clustering results of visual features before making further process for queries and giving different weights to each low-level feature by using OWA operator. The results also show that index structures with OWA adapted outperform the same structures that have equal weighted aggregation in terms of retrieval efficiency, since by using OWA the most relevant visual feature of an object is treated as the main feature (i.e. given the biggest weight) for similarity measurement with a query object. This approach yields better performance according to equal weighted similarity measurement method.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

In this study, we designed and implemented a content-based retrieval system that utilizes Slim-Tree and adapts BitMatrix along with OWA to improve the query performance in terms of time and retrieval efficiency based on low-level descriptors such as color, texture, shape and motion. To the best of our knowledge, BitMatrix is adapted into a content-based retrieval system for efficient video indexing for the first time.

Before low-level feature extraction of videos, by using IBM VideoAnnEx Annotation Tool, sequential video streams are broken into several manageable video shots and a keyframe for each video shot is extracted. The extracted keyframes of the video shots are used for the extraction of still image features of a video shot such as color, texture and shape, and all frames in a video shot are used to extract the motion features of the video shot. To represent the low-level features of video data, we use MPEG-7 descriptors: Color Layout, Dominant Color, Edge Histogram, Region Shape and Motion Activity. These descriptors are extracted by using MPEG-7 XM software and stored in a native XML database, namely Oracle Berkeley DB.

In general, content-based retrieval systems combine features of video data by using fixed weights for each feature. Thus, features of all videos in the database are associated with fixed weights during distance computation. However, when comparing two videos one feature may be more distinctive than other features; therefore that feature must be associated with higher weight. Hence, during distance computations on Slim-Tree and BitMatrix, we use OWA operators to aggregate the distance of each visual feature of any two video shots into a single distance value.

Our system supports exact match, k-NN and range querying of images/video shots by using QBE paradigm. The query performance of the system is tested both on Corel images and on MPEG-7 video test set. From the test results, we observe that BitMatrix with OWA adapted has the best results in terms of time and retrieval efficiency.

Our system does not include the indexing and retrieval of audio objects. Evaluating the retrieval performance of BitMatrix and Slim-Tree index structures of the system for audio objects may be a future direction.

Evaluating the effects of using MPEG-7 motion descriptors other than Motion Activity in the retrieval system is another future direction. The system also does not include face recognition that can be used in content-based image/video retrieval. MPEG-7 provides a face descriptor to retrieve face images similar to a query face image. Thus, another future direction is the integration of face descriptor into the system.

BitMatrix that is adapted to the system is a highly parametrizable index structure. In this study, we use the naïve approach and range expansion heuristic introduced in [1]. Thus, integrating subspace selection approach introduced in [1] and the evaluation of the effects using different cardinality/expansion threshold values on BitMatrix can be stated as future directions. Finally, another future direction may be the parallel processing of BitMatrix index structure by breaking BitMatrix into parts.

REFERENCES

[1] Calistru C., C. Riberio, G. David, "Multidimensional Descriptor Indexing: Exploring the BitMatrix", *pp. 401-410, CIVR'06*, 2006.

[2] Gonalves B., C. Calistru, C. Riberio, G. David, "An Evaluation Framework for Multidimensional Multimedia Descriptor Indexing", ICDE'07 MDDM Workshop, IEEE, 2007.

[3] Traina Jr. C., A. Traina, B. Seeger and C. Faloutsos, "Slim-trees: High Performance Metric Trees Minimizing Overlap Between Nodes", In EDBT 2000, pp. 51-65, Konstanz, Germany, Mar. 2000.

[4] Manjunath B.S., P. Salembier, T. Sikora, "Introduction to MPEG-7:Multimedia Content Description Interface", J.W.& Sons, 2002.

[5] "MPEG-7 Overview (ver. 9)", Int. Org. Stanart., ISO/IEC JTC1/SC29/WG11, May 2003.

[6] Sikora T., "The MPEG-7 Visual Standard for Content Description - An Overview", IEEE Transactions on Circuits and Systems for Video Technology, v. 11, No. 6, June 2001.

[7] Chang S., T. Sikora, and A. Puri, "Overview of the MPEG-7 Standard", IEEE Transactions on Circuits and Systems for Video Technology, v. 11, No. 6, June 2001.

[8] Manjunath B., J. Ohm, V. Vasudevan, A. Yamada, "Color and Texture Descriptors", IEEE Transactions on Circuits and Systems for Video Technology, v.11, No. 6, June 2001.

[9] Bober M., "MPEG-7 Visual Shape Descriptors", IEEE Transactions on Circuits and Systems for Video Technology, v.11, No.6, June 2001.

[10] Jeannin S., A. Divakaran, "MPEG-7 Visual Motion Descriptors", IEEE Transactions on Circuits and Systems for Video Technology, v.11, No.6, June 2001.

[11] Divakaran A., An "Overview of MPEG-7 Motion Descriptors and Their Applications", W. Sharbek, editor, CAIP 2001, Lecture Notes in Computer Science 2124, pages 29-40, Warsaw, Poland, September 2001.

[12] Arslan S., "An XML Based Content-Based Image Retrieval System With Mpeg-7 Descriptors", M.Sc. Thesis, Department of Computer Engineering, METU, Ankara, Turkey, December 2004.

[13] Ciaccia P., M. Patella, and P. Zezula. "M-tree: An efficient access method for similarity search in metric spaces", In Proceedings of the 23rd VLDB International Conference, pp. 426-435, Athens, Greece, August 1997.

[14] Hacid M., C. Decleir, and J. Kouloumdjian, "A Database Approach for Modeling and Querying Video Data", IEEE Transactions on Knowledge and Data Engineering, v. 12, No. 5, September/October 2000.

[15] Ekin A., A. Murat Tekalp, and R. Mehrotra, "Integrated Semantic-Syntactic Video Modeling for Search and Browsing", IEEE Transactions on Multimedia, v. 6, No. 6, December 2004.

[16] Böhm C., S. Berchtold, and D. Keim, "Searching in High-Dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases", ACM Computing Surveys, v. 33, No. 3, pp. 322-373, September 2001.

[17] Chavez E., G. Navarro, R. B. Yates, J. L. Marroquin, "Searching in Metric Spaces", ACM Comp. Surv., Vol. 33, No. 3,, pp. 273–321, 2001.

[18] Gaede V., O. Gunther, "Multidimensional Access Methods", ACM Computing Surveys v.30, No.2, 1998.

[19] Berchtold S., C. Böhm, H. Kriegel, "The Pyramid-Technique: Towards Breaking the Curse of Dimensionality", Proc. Int. Conf. On Management of Data, ACM SIGMOD, 1998.

[20] Lu H., Chin B. C. Ooi, H. T. Shen, and X. Xue, "Hierarchical Indexing Structure for Efficient Similarity Search in Video Retrieval", IEEE Transactions on Knowledge and Data Engineering, v. 18, No. 11, November 2006.

[21] Weber R., H. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces", Proceedings of the 24th VLDB Conference, 1998.

[22] Urruty T., F. Belkouch, C. Djeraba, "KPYR: An Efficient Indexing Method", IEEE, 2005.

[23] Guttman A., "R-Trees: A Dynamic Index Structures for Spatial Searching", ACM, 1984.

[24] Beckmann N., H. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles", ACM, 1990.

[25] Sellis T., N. Roussopoulos, and C. Faloutsos, "The R+-Tree: A Dynamic Index for Multi-dimensional Objects", In Proc. 13th International Conference on Very Large Databases, pp. 507-518, 1987.

[26] Berchtold S., D. Keim, and H. Kriegel, "The X-Tree: An Index Structure for High-Dimensional Data", Proceedings of the 22nd VLDB Conference, 1996.

[27] MPEG-7 XM Homepage, http://www.lis.e-technik.tu-muenchen.de/research/bv/topics/mmdb/mpeg7.html, Last date accessed: September, 2007.

[28] Oracle Berkeley XML DB Homepage, http://www.oracle.com/database/berkeley-db.html, Last date accessed: January, 2008.

[29] XXL Homepage, http://dbs.mathematik.uni-marburg.de/?search=Research_Projects_XXL, Last date accessed: January, 2008.

[30] Weka Homepage, http://www.cs.waikato.ac.nz/ml/weka/, Last date accessed: January, 2008.

[31] Colt Project Homepage, http://dsd.lbl.gov/~hoschek/colt/, Last date accessed: January, 2008.

[32] VideoAnnEx: IBM Video Annotation Tool, http://www.research.ibm.com/VideoAnnEx/, Last date accessed: October, 2007.

[33] Eidenberger H., "How good are the visual MPEG-7 features", Proc. of the 5th ACM SIGMM Int. WS on Mm. info. retrieval, pp.130-137, Berkeley, 2003.

[34] Guner K. K., "MPEG-7 Compliant ORDBMS Based Image Storage and Retrieval System", MS Thesis, Middle East Technical University, January 2004.

[35] Yager R.R., "On ordered weighted averaging aggregation operators in multi-criteria decision making", IEEE Trans. Sys. Man Cyb. 18, pp. 183-190, 1988.

[36] Yoshitaka A., T. Ichikawa, "A Survey on Content-Based Retrieval for Multimedia Databases", IEEE Transactions on Knowledge and Data Engineering, v.11 No.1, 1999.

[37] MUVIS Project Homepage, http://muvis.cs.tut.fi/, Last date accessed: January, 2008.

[38] Zhang D., G. Lu, "Evaluation of Similarity Measurement for Image Retrieval", IEEE International Conference Neural Networks & Signal Processing, December 2003.

[39] Gonzalez R., R. Woods, "Digital Image Processing", Prentice-Hall, 2002.

[40] WebSEEk Homepage, http://persia.ee.columbia.edu:8008/, Last date accessed: January, 2008.

[41] Lee J., H. Kim, and W. Kim, "Video/Image Retrieval System based on MPEG-7", IEEE, 2003.

[42] Esen E., Ö. Önür, M. Soysal, Y. Yasaroglu, S. Tekinalp, and A. Aydin Alatan, "A MPEG-7 compliant Video Management System: BilVMS", Proc. of 4th European Workshop on Image Analysis for Multimedia Interactive Services, April 2003.

[43] Hampapur A., A. Gupta, B. Horowitz, C. Shu, C. Fuller, J. Bach, M. Gorkani, R. Jain, "Virage Video Engine", Proc. SPIE Vol. 3022, p. 188-198, January 1997.

[44] Flickner M., H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker, "Query by Image and Video Content: The QBIC System", IEEE, 1995.

[45] Lee J., J. Oh, S. Hwang, "STRG-Index: Spatio-Temporal Region Graph Indexing for Large Video Databases", ACM SIGMOD, June 2005.

[46] Corel Database, http://www.corel.com, Last date accessed: January, 2008.