Efficient and Effective Subimage Similarity Matching for Large Image Databases

by

Kai Sang Leung

B.Sc. (Honours), The University of British Columbia, 1995

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

August 1997

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science
The University of British Columbia
2366 Main Mall
Vancouver BC
Canada V6T 1Z4

Date: August 30, 1997

Abstract

As network connectivity has continued its explosive growth and storage devices have become smaller, faster, and less expensive, the number of on-line digital images has increased rapidly. Correspondingly, efficient and effective content-based retrieval systems for handling image queries have become necessary. In addition, users are often interested in local contents within subimages. In this thesis, we develop Padding and Reduction Algorithms to support subimage queries of arbitrary size based on local color information. The idea is to estimate the best-case lower bound to the dissimilarity measure between the query and the image. By making use of multiscale representation, this lower bound becomes tighter as the scale becomes finer. Because image contents are usually pre-extracted and stored, a key issue is how to determine the number of levels used in the representation. We address this issue analytically by estimating the required CPU and I/O costs, and experimentally by comparing the performance and the accuracy of the outcomes of various filtering schemes. Our findings suggest that a 3-level hierarchy is preferred.

We also study three strategies for searching multiple scales. Our studies indicate that the hybrid strategy with horizontal filtering on the coarse level and vertical filtering on remaining levels is the best choice when using Padding and Reduction Algorithms. Using the hybrid search strategy in the multiscale representation with the determined number of levels, the best 10 desired images can be retrieved efficiently and effectively from a collection of a thousand images in about 3.5 seconds.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to sincerely thank Dr Raymond Ng, my supervisor, for his support and guidance throughout the course of my work. His advice, comments, and suggestions concerning this research project were invaluable.

I would also like to thank Dr Jim Little, my second reader, for the time that he spent reading and reviewing my work. His comments and recommendations were very helpful and valuable.

I am indebted to my parents for their support and love. Thanks to friends as well as faculty members, staffs, and fellow students in the Department for their help and encouragements; thanks also to the Natural Sciences and Engineering Research Council of Canada for the financial support in the form of a Postgraduate Scholarship.

KAI SANG LEUNG

The University of British Columbia

August 1997

To my parents

Chapter 1

Introduction

1.1 Image Databases

Over the past few decades, database management systems (DBMSs) have been recognized as tools with practical value for handling large amounts of data. A primary purpose of a DBMS is to provide an environment for efficient and effective retrieval and storage of data. Until recently, many DBMSs were designed to handle only alphanumeric data.

In recent years, advances in technologies for scanning, networking, and CD-ROMs have lowered the prices for disk storage. These advances, coupled with acceptance of common image compression and file formats, enable users to acquire, store, manipulate and transmit large numbers of images. As a result, the number of digital image archives has increased tremendously. The use of images and graphics in World-Wide Web publications has also increased at an incredible rate. In addition, images are generated at an increasing rate by growing number of sources such as civilian and military satellites as well as commercial satellites, instruments used in petroleum and mining industries, medical information systems, Geographic Information Systems (GIS), and art galleries and museum management systems. For example, an estimated 281 gigabytes (GB) of image data will be produced daily by the instruments on two Earth Observing Systems plat-

forms [Castelli *et al* 1997]. With all these changes, we need the DBMSs to support not only traditional alphanumeric data, but also visual-based data in the form of still images.

Image databases also play important roles in the four "Grand Challenge" application domains — (1) Geographic/Environmental Information Systems, (2) Engineering/Scientific Visualization Systems, (3) Medical Information Systems, and (4) Education Systems — which were identified in the 1992 US National Science Foundation workshop on visual information management systems [Jain 1993]. The demands for image database management systems (IDBMSs) also increase in other application areas like stock-photography for remote printing, retail cataloging, art work retrievals, advertisement creation, and imaging clip arts [Sawhney and Hafner 1993, Barber *et al* 1994, Gudivada and Raghavan 1995, Petkovic *et al* 1996, Castelli *et al* 1997].

1.2 Content-Based Retrieval

Traditional database retrieval technology uses an "exact match" approach. To do so, traditional structured alphanumeric information is indexed by descriptive keywords or numeric descriptors. Due to the success in handling alphanumeric data, many IDBMSs also use the "exact match" approach in handling image data. In particular, images are indexed based on some identifying text such as titles, captions, creator's names, production years, or catalog numbers. However, these identifiers are usually "external" to image contents (like color, texture, and shapes). Searching these external features of the images in the collection may not necessarily yield fruitful results. In many situations, users may remember the contents, but not the associated identifying text, of the images they have seen before. Thus, retrieval methods based on "external" identifiers may not be helpful. For example, if the images in a collection are indexed by names of painters, a user may not be able to retrieve the image of Mona Lisa unless he remembers the painter's name, Leonardo da Vinci.

Complementary to retrieval methods based on "external" identifiers, content-based retrieval (CBR) methods have been developed. With CBR, images are searched and indexed based on contents of the images. These contents can be classified into two main types:

1. "syntactic" contents — such as color, texture, and shape — which are context independent, and

2. "semantic" contents — such as objects — which are context dependent.

To exploit existing advantages of the "exact match" approach, some CBR systems rely on manually associating textual descriptions with the image contents. However, these textual content descriptors are subjective, domain dependent, and expensive to produce. Sometimes, the image contents of the underlying data, like texture and shape, are difficult or nearly impossible to describe with text annotations. It is well known that "an image is worth a thousand words"; descriptors/keywords are often incomplete and not precise enough for satisfactory image retrievals. For example, an indexer may assign keyword "sunrise" to an image which the user may perceive to be a sunset. As another example, for a picture taken at dawn, an indexer may associate the text annotation "moon" with an object which the user may perceive to be the sun.

Given the rapid growth in the availability and demand for image data, the high labor cost involved with manually associating text annotations with images, and the difficulty of anticipating every user's needs when assigning keywords and descriptors, it is more efficient to represent the image contents, particularly "syntactic" contents, by automatically extracted features. Since the "syntactic" contents are context independent, they can be extracted without prior knowledge of the images. In many CBR systems, these contents are extracted automatically using image processing methods. The results are feature vectors representing the contents. Examples of these vectors include:

- color

  Color can be represented by three-dimensional average color histograms [Niblack *et al* 1993, Sawhney and Hafner 1993, Faloutsos *et al* 1994, Ashley *et al* 1995, Flickner *et al* 1995], low-dimensional dominant color histograms (sets of the most frequently occurring colors) [Ashley *et al* 1995], or three-dimensional moment-based color distributions composed of the first 3 moments (average, variance, and skewness of each color) [Stricker and Orengo 1995, Dimai and Stricker 1996]. Since the discriminating power with these low dimensional feature vectors may not be adequate in some applications, color is also represented by n-dimensional (where $n \gg 3$, say 64) color histograms in other applications [Ioka 1989, Niblack *et al* 1993, Sawhney and Hafner 1993, Faloutsos *et al* 1994, Sawhney and Hafner 1994, Ashley *et al* 1995, Flickner *et al* 1995, Tam 1996].

- texture

  Retrieval is often based on modified versions of coarseness, contrast, and directionality [Niblack *et al* 1993, Ashley *et al* 1995]. Coarseness measures the scale of the texture — the average size of the regions of similar intensity in an image; contrast describes the vividness of the pattern — the amount of variation between the light and dark areas in an image; and, directionality describes whether the image has a favored direction or is isotropic.

- shape

  Shape can be represented in several different ways. A common representation is 20-element shape vectors composed of a combination of heuristic shape features (for example, area, circularity, eccentricity, and major axis orientation) and sets of algebraic moment invariants [Niblack *et al* 1993, Ashley *et al* 1995].

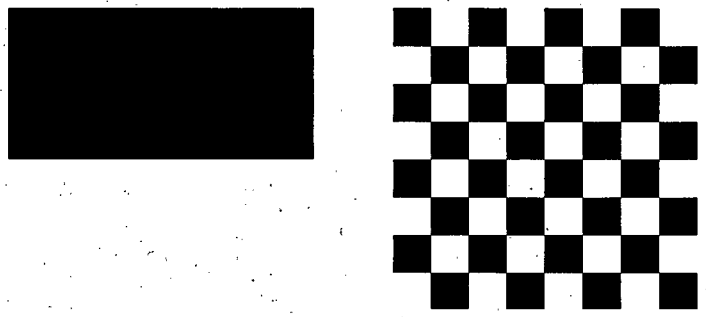
Regarding the "semantic" contents, they are context dependent and need to be extracted with prior knowledge of the images and of real world properties. Automatic extraction of the contents is not easy, and usually requires some computationally expensive scene analysis methods.

1.2.1 Color and Spatial Similarity Matching

Demands for Color and Spatial Similarity

In the vision and image processing community, color has been widely used for image segmentation and classification tasks. Studies have been done suggesting that color plays an important role in human understanding of an image [Hurvich 1981]. This explains why color is one of the most popular among all image contents, and is supported by many IDBMSs like QBIC [Flickner *et al* 1995], Virage [Bach *et al* 1996, Gupta 1997], and Miyabi [Hirata *et al* 1993].

A simple way to represent color is to use a single global three-dimensional average color histogram or a single global n-dimensional color histogram. Similarity matching is then performed by comparing the global feature vector of the query with the global feature vector of each image in the database. Matching appears to be computationally efficient. Color histograms contain information about the statistical distribution of various colors within the image; however, they lack information about the spatial locations of different colors. As a result, CBR without spatial information of color may not be effective due to the "random effect". For example, the color histogram for an image with upper portion black and lower portion white is the same as the color histogram for black/white checkerboard (Figure 1.1). Hence, CBR with both color and spatial information provides better selectivity.



50% black
50% white
(a) Black/White Block

50% black
50% white
(b) Black/White Checkerboard

Figure 1.1: Example of "Random Effect"

Processing of Color and Spatial Similarity

Regarding the processing of spatial similarity matching, a method for handling spatial matching of objects exists [Sistla *et al* 1994, Sistla *et al* 1995]. In their system, the authors use meta-data to describe the spatial relationships of objects. The meta-data are generated *a priori* and stored. During the query processing step, new rules about the spatial relationships of objects can be deduced from the meta-data. With deductive reasoning about spatial relationships of objects, desired images can then be retrieved. This method handles spatial matching of objects quite well, but it may not be practical in handling spatial matching of color because meta-data on color cannot be generated in the same way as the meta-data on objects. For example, given a checkerboard, it is not easy to describe the spatial relationships of the colors "black" and "white".

A natural method to handle similarity matching with color and spatial information is direct spatial comparisons of the pixel colors of the query and the images. However, it can be more computationally expensive than comparison of global feature vectors because the number of pixels in an image is usually much greater than the number of dimensions in the feature vector. Moreover, it is usually assumed that the query and images are properly

aligned in the direct spatial comparison; this assumption may not hold in many cases as the user may remember only an approximate location of color when specifying the query. With these problems, direct spatial comparison may not be helpful.

An alternative method to capture information about the spatial distribution of color is to divide an image into several blocks and create a color histogram for each of the blocks. For example, the system presented by Gong *et al* divides the image into 9 (= 3×3) blocks [Gong *et al* 1994]. In QBIC, the image is divided into a grid of either (a) 6 vertical $\times$ 8 horizontal blocks or (b) 9 vertical $\times$ 12 horizontal blocks for partition-based search [Ashley *et al* 1995]. In these systems, each image block is of a certain size and of a certain scale. So, similarity matching is applied on these uni-scale image blocks to handle color and spatial similarity. To improve the efficiency and the effectiveness of CBR, it seems appropriate to explore and analyze the possible use of multiscale matching.

1.2.2 Multiscale Matching

Demands for Multiscale Matching

For similarity matching with fixed grid segmentation, color histograms for all blocks in the grid need to be examined during the query processing step. The efficiency and the effectiveness of CBR systems depend on the quality of the segmentation and the relevance of the segmentation to the user's needs. In some situations, the scale at which the images are blocked may be considered too fine. Applying similarity comparisons to all these fine blocks of the query and the images may not be worth the effort. For example, comparing every block of a red-only query with the corresponding block of the non-red images seems inefficient. In other situations, the scale at which the images are divided may be considered not fine enough for discriminating the desired images from the collection. Hence, picking the best scale at which the images are blocked is not easy.

To deal with the problem, multiscale matching can be applied. Poor matches can be

identified and then eliminated by comparisons on feature vectors at the coarse granularity level. For example, non-red images can be filtered out by comparing global feature vectors of the red-only query and the database images. On the other hand, if the current scale does not provide satisfactory discriminating power, matching can be performed at a finer granularity level when using multiscale matching.

Processing of Multiscale Matching

To aim for efficient and effective CBR, a search algorithm that makes use of multiresolution wavelet decompositions of query and images has been proposed [Jacobs *et al* 1995]. Coefficients of the decompositions are distilled through processes of truncation and quantization. During the query processing step, the algorithm simply compares the number of distilled coefficients that are common in both the query and the images. With the algorithm, desired images can be retrieved efficiently and effectively by submitting a whole-image query. To do so, users are required to know the color distribution of the entire image. In many cases, users may not know or care about the color distribution on some portion of the image, and they may want to retrieve the desired images by posing a subimage query. Unfortunately, subimage queries cannot be handled by the algorithm.

Recently, a formal framework was presented for designing search algorithms which can identify target images by spatial distribution of color [Chen *et al* 1997]. The framework is based on a multiscale representation of both the image data and the associated parameter space that must be searched. To process the whole-image query, a branch-and-bound algorithm is used. The algorithm eliminates poor matches at coarse scales with minimum computation. This insures that each image is only searched to a scale necessary to determine if it is a potential match candidate.

Using the branch-and-bound algorithm, all the images in the database are checked at a coarse scale in the first iteration. Then, the algorithm proceeds to finer scales in

non-descending order of distance value [1]. In general, the branch-and-bound algorithm works in such a way that it always keeps track of distance values of all images contending for further consideration. Images with smallest values are "extended" one level (to the next/finer scale). Then, these most recently "extended" images are considered, along with the remaining old ones; again, images with smallest values are "extended". The process repeats until the target images are found. Notice that at each iteration, the next set of images to be "extended" contains images with smallest values, and the set does not necessarily include images in the current set or images at the current level. Hence, jumping back and forth among images and levels may occur frequently. For large image databases, such jumping makes it hard to optimize file organization and buffer management, and may impose a high I/O cost. To avoid jumping, developments of some other search strategies seem appropriate.

Like similarity matching in the uni-scale representation, color histograms at each scale are usually extracted and stored prior to execution time. The efficiency and the effectiveness of the CBR systems are expected to be affected by the number of levels used; as such, we need to choose the number of levels carefully through analyses and experiments. However, detailed analytical and experimental results for the determination of a suitable number of levels are seldom provided in depth.

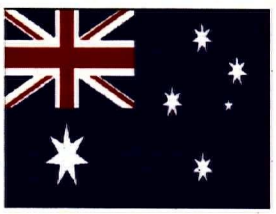
Furthermore, it is observed that these multiscale systems cannot handle subimage queries. As subimage queries are useful in many applications, investigations on how to make use of multiscale representation in developing algorithms for dealing with subimage queries are worth pursuing.

[1] A distance value measures the dissimilarity/distance between the query and the image.

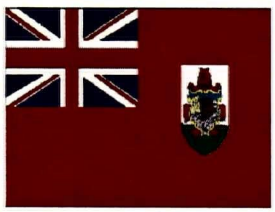
1.2.3 Subimage Query Matching

Demands for Subimage Queries

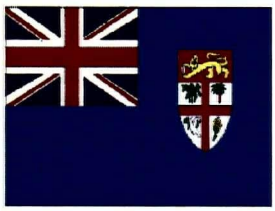
Many current IDBMSs support only whole-image matching, but not subimage matching. However, it is well known that human memory is weak in retaining a fine granularity of spatial information of color. In many cases, the user does not remember every detail of the image he has seen before, and he remembers only a portion of the image. And in other cases, the user is interested in local image contents. For example, a user may want to find images of national flags with the Union Jack in the upper left corner. As another example, a user may be interested in finding all images of sunrises where the sun is rising in the upper right portion of the image. Then, whole-image matching may not be helpful because the user does not know or care about the color distribution on the remaining portion of the image. For instance, each of the national flags in Figure 1.2 contains the Union Jack in the upper left corner, but the remaining portion is quite different. With whole-image matching, the user needs to come up with the color distribution on the remaining portion of the image. Hence, subimage query matching is needed.



upper-left: Union Jack
remaining: mainly blue
(a) Flag of Australia



upper-left: Union Jack
remaining: mainly red
(b) Flag of Bermuda



upper-left: Union Jack
remaining: mainly light blue
(c) Flag of Fiji

Figure 1.2: National Flags

Processing of Subimage Queries

As mentioned in Subsection 1.2.1, in order to capture information about the spatial distribution of color, some IDBMSs divide an image into several blocks and create a histogram for each of the blocks. It is observed that a subimage query may occupy some but not all the blocks. So, to handle subimage matching, one can compare only the histograms of the "occupied" blocks of the query and the corresponding blocks of the images. For the systems in which the user can assign weights to different blocks, one can assign a weight of 0 to the "vacant" blocks (blocks that are not occupied by the subimage query). However, a problem with these techniques is that it is not unusual to have a query whose size is not an integral multiple of the chosen block size (for example, a subimage query — whose size is $\frac{1}{16}$ of the image — can be submitted to a collection of images with 2×2 segmentation, in which, each segment covers $\frac{1}{4}$ of the entire image).

To deal with subimage queries of arbitrary size, template matching algorithms can be applied. The idea is that the user query consisting of $s \times s$ pixels is served as a "template" and is compared with every image subregion of the same size. Notice that if an image contains $S \times S$ pixels (where $S \geq s$), there are $(S - s + 1)^2$ image subregions having the same size as the "template". The smaller the query size, the greater the number of image subregions to be compared. This may lead to a huge amount of computation, and thus algorithms based on template matching are usually inefficient. In general, each image contains S^2 subregions of size 1×1 pixel, $(S-1)^2$ subregions of size 2×2 pixels, and so on. These add up to a total of $\frac{S(S+1)(2S+1)}{6}$ subregions with size ranging from 1×1 pixel to $S \times S$ pixels. Thus, pre-extraction becomes impractical. For these template matching algorithms, image contents are usually not precomputed or stored, and execution times are expected to be very high. Therefore, other algorithms for performing subimage matching efficiently and effectively are needed.

1.3 Problem Definition and Contributions

In the previous section, the demands for similarity matching on color and spatial information have been mentioned. To process the query, some IDBMSs use fixed grid segmentation approaches. Some multiscale matching approaches have been proposed to further improve the efficiency and the effectiveness of CBR. However, detailed analytical and experimental results on the determination of the suitable number of levels for the approaches are seldom reported, and comparisons of different search strategies using multiscale representation are also rare. Moreover, subimage queries, which are in demand, cannot be handled by many IDBMSs. Techniques for handling these subimage queries of arbitrary size are needed. Therefore, in this thesis, the following questions are investigated:

1. How can we deal with subimage queries of arbitrary size ?

   Two efficient and effective algorithms for handling subimage queries are developed, namely Padding and Reduction Algorithms. The idea is that Padding and Reduction Algorithms estimate the best possible color histograms for a subimage query $\mathtt{Q}$ and an image block $\mathtt{I}$ of size larger than $\mathtt{Q}$ by either:

   (a) enlarging the query $\mathtt{Q}$ into a new query $\mathtt{Q}^{\text{Enlarged}}$ that is of the same size as the image block $\mathtt{I}$, or

   (b) reducing the image block $\mathtt{I}$ to a new block $\mathtt{I}^{\text{Reduced}}$ that is of the same size as the query $\mathtt{Q}$.

   More precisely, Padding and Reduction Algorithms give lower bounds to the Euclidean distance between the histograms of subimage query $\mathtt{Q}$ and image block $\mathtt{I}$:

$$\text{Padding:} \quad \min\ (\mathtt{Q}^{\text{Enlarged}}_{\text{Histogram}} - \mathtt{I}_{\text{Histogram}})^T\ (\mathtt{Q}^{\text{Enlarged}}_{\text{Histogram}} - \mathtt{I}_{\text{Histogram}})$$

$$\text{Reduction:} \quad \min\ (\mathtt{Q}_{\text{Histogram}} - \mathtt{I}^{\text{Reduced}}_{\text{Histogram}})^T\ (\mathtt{Q}_{\text{Histogram}} - \mathtt{I}^{\text{Reduced}}_{\text{Histogram}})$$

For given Q and I, both algorithms provide the same best-case lower bound distance estimation. However, their efficiency may differ significantly, depending on the size differential between Q and I. If the size differential is large, the Reduction Algorithm is more efficient; otherwise, the Padding Algorithm is the winner.

Depending on the location of I, similarity matching can incorporate color similarity as well as spatial similarity. For instance, the distance between Q and I is defined as a weighted sum of the form:

$$\beta \times \text{histogram distance} \; + \; (1-\beta) \times \text{positional distance}$$

where the histogram distance is computed by either the Padding Algorithm or the Reduction Algorithm. The weighting factor β is chosen by the user to specify a preference on the relative importance of color distribution and its spatial information.

2. How many levels do we need for multiscale representation ?

With multiscale representation, an image is divided into several blocks, and an associated color histogram is created and stored for each of the blocks. Each block can be further divided into subblocks. In this thesis, images are divided into four levels of (sub)blocks:

(a) At Level H, the entire image is represented by a single color histogram.

(b) At Level I, the image is divided into four (non-overlapping) blocks [2], and each block is represented by a color histogram covering $\frac{1}{4}$ of the entire image.

(c) At Level J, each block at Level I is further divided into four subblocks, and each subblock is represented by a color histogram covering $\frac{1}{16}$ of the entire image.

(d) At Level K, each subblock at Level J is again divided into four, each of which is represented by a color histogram covering $\frac{1}{64}$ of the entire image.

[2] Alternatively, an image in the database can be divided into five or nine overlapping blocks.

Given user queries of arbitrary size, we investigate the number of levels that are required for efficient and effective retrieval of the desired images. With four levels of (sub)blocks for each image, many multi-level filtering schemes can be developed to find the desired matches. For example, the choices include a four-level HIJK scheme, a one-level J scheme, a two-level HJ scheme that skips Level I, and some other schemes. Analytically, we set up cost models to evaluate all 15 possible filtering schemes, and estimate their CPU, I/O, and combined CPU&I/O costs. The results tend to favor:

- the filtering schemes which start with Level H (or Level I), and
- the filtering schemes which do not skip the intermediate level.

Experimentally, given subimage queries of particular size, both the efficiency (for example, the time required to find the desired images) and the effectiveness (for example, the number of desired images being retrieved) are measured. The results show that the three-level HIJ filtering scheme works well for most queries when operating together with the Padding and Reduction Algorithms mentioned earlier. Therefore, only three levels (up to 4×4 segmentation) are needed for the multiscale representation.

3. What is the best strategy for searching multiple scales ?

The branch-and-bound algorithm proposed by Chen *et al* [Chen *et al* 1997] is accurate in its search, but it can be inefficient for large databases. In particular, the number of images the algorithm must keep track of can be large, and jumping back and forth among images and levels may be required frequently. For large databases, such jumping makes it hard to optimize file organization and buffer management, and may generate a large number of I/Os.

In this thesis, we consider three strategies that try to avoid the kind of jumping mentioned above:

(a) Search PV (Pure Vertical) — a strategy in which a vertical filter searches each image "vertically" across scales.

The idea is that images are checked one after another. At any point in time, a set of u images currently having smallest distance values are kept (where u is the number of images requested by the user). For each image, the algorithm keeps proceeding to finer scales until (i) the distance value at a particular scale is already so large that the image cannot be qualified as a good match, or (ii) the finest scale is reached and the image is either discarded or selected as a member of the answer set, depending on the distance value.

(b) Search PH (Pure Horizontal) — a strategy in which horizontal filters search "horizontally" across the database level by level.

The idea is that all the images in the database are checked by the filter at the coarse scale in the first iteration. Poor matches at this scale are eliminated. In subsequent iterations, images that are left from the previous level/scale are checked by filters at finer scales, and poor matches are again removed. The process repeats until the finest scale is reached and the top u images are returned.

Search PH is more efficient than Search PV, because the latter tends to require many comparisons at the finest scale, particularly at the beginning of the search. However, in Search PH, it is hard to determine the number of images to be carried over from the current level to the next level. For some queries, if such a number is small, an image `I` that gives a good match at a finer scale may have been eliminated before reaching the finer scale. This may happen when there are sufficiently many images which are not as good as `I` at

the finer scale but are better than I at the coarser scale. Consequently, while delivering efficiency, Search PH may suffer from a loss of effectiveness.

(c) Search HV (Horizontal-and-Vertical) — a hybrid of the above two search strategies in which we use a horizontal filter on the first level and vertical filters on the remaining levels.

The idea is that all the images in the database are checked by a horizontal filter at the coarse scale in the first iteration. Poor matches at the coarse scale are eliminated. For each of the images that are left, the algorithm keeps proceeding to finer scales until (i) the distance value at a particular scale is already so large that the image cannot qualify as a good match, or (ii) the finest scale is reached and the image is either discarded or selected as a member of the answer set, depending on the distance value.

In this strategy, the detailed search with the use of vertical filters is applied not to the set of all the images, but only to its most promising subset. The experimental results confirm that Search HV is the best strategy for retrieval of desired images when compared with the other two strategies because Search HV keeps a good balance of performance/speed and accuracy.

1.4 Outline of Thesis

In the next chapter, related works are described. We give an overview of some of the research projects in CBR, and mention the research projects for handling color and spatial similarity, multiscale matching, and subimage queries. Chapter 3 focuses on Padding and Reduction Algorithms. Concepts, implementations, and experimental results are discussed. Chapter 4 describes the multiscale representation and a strategy for searching such a representation (Search PV). Analytical and experimental results for each filtering

scheme are studied so that the second question mentioned in Section 1.3 can be answered. In Chapter 5, two more search strategies, namely Search PH and Search HV, are proposed. We compare both analytical and experimental results of the three search strategies. As a result, an answer to the third question mentioned in Section 1.3 is provided. Finally, conclusions and suggestions for future work are presented in Chapter 6.

Chapter 2

Related Works

Efficient and effective content-based retrieval systems usually require that several important objectives are satisfied, namely:

- efficient indexing methods,
- expressive query language, and
- efficient and effective query processing and optimization.

To achieve the goals, several research projects have been carried out.

2.1 Indexing

To handle a user query, image contents of the query are compared to the corresponding image contents of the images in the database to determine which images are good matches. For efficiency, image contents are usually captured by feature vectors which are precomputed and stored. For a small database, sequential scanning of these feature vectors is fast. However, as the database grows, the linear scale-up of the sequential scanning becomes prohibitively slow. One way to speed up the process is to treat each feature vector as a point in n-dimensional space, and employ multi-dimensional indexing structures. Many multi-dimensional indexing structures have been designed. Examples are KD-trees [Samet 1990], R-trees [Guttman 1984], R^+-trees [Sellis *et al* 1987], R^*-trees [Beckmann *et al* 1990], SS-trees [White and Jain 1996], SS^+-trees [Kurniawati *et al* 1997], TV-trees [Lin *et al* 1994], VP-trees [Chiueh 1994], and X-trees [Berchtold *et al* 1996].

The above indexing structures help in improving the efficiency for low dimensional feature vectors. However, for high dimensions, many of these multi-dimensional indexing structures explode exponentially with the dimensionality, and eventually reducing to sequential scanning. Given that image feature vectors are usually of high dimensionality (for example, it is not unusual to have 64-dimensional vectors for color features), ways to deal with this problem are needed. One way to deal with this "dimensionality curse" problem is by using filters. A 2-level filtering approach has been proposed [Sawhney and Hafner 1993, Faloutsos *et al* 1994, Sawhney and Hafner 1994]. The idea is to abstract a low dimensional vector from each original high dimensional feature vector by an information preserving transformation. Based on distance values of the abstracted vectors, images which are far from the query are eliminated. As a result, only a small number of candidates are left. The original high dimensional vectors of these candidates are then passed to the detailed matching operation to obtain the best matches. Unfortunately, the dimensions of abstracted vectors cannot be too high; otherwise, vectors cannot take full

advantage of multi-dimensional indexing structures. On the other hand, the dimensions of abstracted vectors cannot be too low; otherwise, the detailed matching at the finest level needs to be operated on a large number of high dimensional vectors. To improve the efficiency, a 3-level filtering approach has been proposed [Tam 1996, Ng and Tam 1997]. The idea is to add an additional intermediate level so that both the coarsest and the finest filterings can be more efficient.

As the efficiency of spatial indexing structures usually deteriorates with the increase in dimensionality, methods to compress or reduce the dimensionality of the image vector space without losing much information are necessary. One method is Karhunen-Loeve (KL) Transformation, or Principal Component Analysis [Sedighian 1995, Ng and Sedighian 1996]. The idea is to transform the data space by removing dependent dimensions and converging most of the information into the first few dimensions. Another method is to apply singular value decomposition and clustering techniques recursively to feature vectors until the dimensions cannot be further reduced [Thomasian *et al* 1997].

2.2 Query Language

With the ever-growing use of the Internet, there are more and more image database servers. To maximize the efficiency and the throughput of an image database server, it is beneficial to have an expressive query language so that the user can be as precise as possible in specifying his query. Such precision in query specification may lead to reductions in the number of query reformulations over the network. In the past few years, many query language systems have been developed. Examples are QBIC [Sawhney and Hafner 1993, Faloutsos *et al* 1994, Sawhney and Hafner 1994, Flickner *et al* 1995, Finn 1996], Virage [Bach *et al* 1996, Gupta 1997], Miyabi [Hirata *et al* 1993], Photobook [Pentland *et al* 1994], and FMR [Sakamoto *et al* 1994]. With these systems, querying from image databases for applications with a dense image space (for example, medical image management

or remote-sensing where the images are very similar) as well as a sparse image space (for example, art gallery management where the neighboring images are different from each other) is possible. Recently, another query language, EXQUISI, has been proposed [Faulus 1996, Faulus and Ng 1996, Faulus and Ng 1997]. To incorporate imprecision and ambiguities in user queries, the user is allowed to specify a range of values for image content, and all values within the range are then treated as identical during the query processing step. Moreover, an additional reformulation function is provided so that the user can specify the parts of the returned image he wants to include or exclude.

2.3 Query Processing based on Color and Spatial Similarity

In the vision and image processing community, color has been widely used for image segmentation and classification tasks. Studies have been done suggesting that color plays an important role in human understanding of an image [Hurvich 1981]. This explains why, among all image contents, color is one of the most popular, and is supported by IDBMSs like QBIC [Flickner *et al* 1995], Virage [Bach *et al* 1996, Gupta 1997], and Miyabi [Hirata *et al* 1993].

A popular way to represent color is to use color histograms. A color histogram holds information on color distribution, but it lacks spatial information on color. The problem may be overcome by dividing an image into several blocks and creating a histogram for each of the blocks. Locality information is captured as each of the histograms holds color distribution for a particular block in the image. The more blocks in the image, the more accurate is the locality information; however, more memory would be consumed in holding the histograms, and more computation time would be required for comparing histograms. It is a tradeoff between efficiency (time and space efficiency) and effectiveness.

In QBIC, the user is allowed to submit queries on large image databases based on example images (query by example) as well as user-constructed sketches and draw-

ings (query by painting) [Sawhney and Hafner 1993, Faloutsos *et al* 1994, Sawhney and Hafner 1994, Ashley *et al* 1995, Flickner *et al* 1995, Finn 1996]. For query by example, similarity matching between the histogram of a query and the histogram of an image is based on the weighted Euclidean distance of the normalized histograms:

$$(\text{Query}_{\text{Histogram}} - \text{Image}_{\text{Histogram}})^T A\, (\text{Query}_{\text{Histogram}} - \text{Image}_{\text{Histogram}})$$

where A is a similarity matrix with entries a_{ij} describing similarity between color i and color j. The similarity matrix accounts for both the perceptual distance between the pairs of colors and the difference in the amounts of each color. Two types of histograms are used:

1. average Munsell color histograms for handling average color queries. This kind of histogram is useful for images that have a dominant color or a small range of hues. The 3-dimensional average color histogram for each image is formed by adding up the red, green and blue components of each pixel.

2. n-dimensional color histograms for handling histogram color queries. This kind of histogram is useful for searching images with a desired color distribution. To create a histogram, color space is usually divided into 64 ranges, and the percentages of pixels in each color range are counted. As a result, a 64-dimensional color histogram is formed.

Regarding query by painting, QBIC retrieves images with similar colors in similar spatial arrangement. For a partition-based method [Ashley *et al* 1995], each image in the database is divided into a grid of either (a) 6 vertical $\times$ 8 horizontal blocks or (b) 9 vertical $\times$ 12 horizontal blocks. For each block, (a) an average Munsell color histogram and (b) a partial color histogram consisting of the most frequently occurring colors and their frequencies are computed and stored. At runtime, image contents of the query are ex-

tracted in a similar manner. Similarity matching is based on the average of the distance values of all the blocks.

Some other methods for representing color and spatial information and for computing similarity measures have been suggested. For example, in the system presented by Gong *et al* [Gong *et al* 1994], the image is divided into 3×3 subareas. In addition to a histogram for the entire image, a histogram is created for each of the 9 subareas. With the observation that colors within an image do not spread widely in the color space, the authors use the top 20 bins (in terms of pixel counts) of the color histogram. Two parameters, the Weighted Perimeter and the Weighted Angle, are extracted from the top 20 bins; they form hyper-polygons. Similarity matching is computed by comparing the values of the two parameters.

In another system [Stricker and Orengo 1995, Dimai and Stricker 1996, Stricker and Dimai 1996] — this one based on the observation that important objects are usually placed in the center of images in many applications — the image is divided into five fuzzy regions (Figure 2.1): center, top-left, top-right, bottom-left, and bottom-right regions. The authors use moment-based color distributions — (1) average, (2) variance, and (3) skewness of each ($L^*a^*b^*$) color channel — which they claim to be more robust and more efficient than working with color histograms.

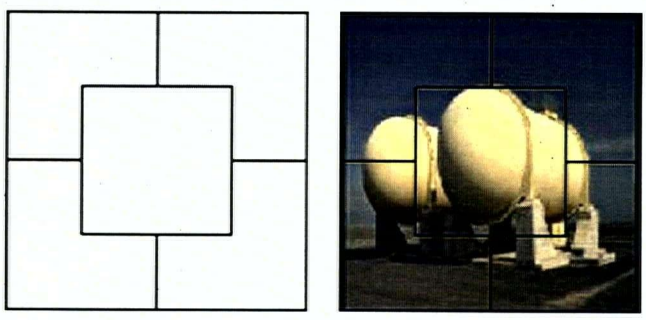


Figure 2.1: Fuzzy Regions

In Miyabi [Hirata *et al* 1993], images are divided into 8 regions using values on color and other image contents (such as texture). With an image regioning and merging technique, color information is encoded into a picture index and can be used in matching.

For the VisualSEEk system [Smith and Chang 1996], Smith and Chang notice the difficulty of picking the best scale at which the images should be blocked in fixed block segmentation. They use the color histogram back-projection method of Swain and Ballard [Swain and Ballard 1990] to segment the images instead. The encoding of color information is done by using a binary color set in which only the colors that are sufficiently present in the region are selected. With this color set back-projection method, images can be retrieved effectively.

One common point in these systems is that they have not explored the use of multiscale matching for further improvements on the efficiency and the effectiveness of CBR based on color and spatial similarity.

2.4 Query Processing with Multiscale Matching

The difficulty of picking the best scale at which the images should be divided in an IDBMS can be addressed by the use of multiscale representation. With the representation, comparing feature vectors at a coarse granularity level enables the identifications of poor matches from the collection of images. On the other hand, if satisfactory discriminating power cannot be provided by the current scale, matching can be done at a finer granularity level which may lead to the retrieval of the desired images.

In QBIC, query by painting is not only handled by a partition-based method, but also by a region-based method [Ashley *et al* 1995]. Instead of relying on a fixed grid placed on the image, an approximate segmentation is used on the images and the query. The iterative metric space clustering algorithm [1] is used, which starts with each color in the

[1] The algorithm is based on the concept of mutual nearest neighborhood.

image defining a cluster. At each iteration, a pair of clusters is collapsed into a single cluster if their mutual ranks fall below the preset threshold. Color distance and spatial distance are combined during the clustering stage. For each color, a bounding rectangle is formed for each group of connected pixels having that color. Then, the bounding rectangles for a given color are successively clustered until one rectangle remains. As a result, multi-level color trees are formed. To process a user query, the image contents of the query are matched against the image contents of images in the database. In order to do so, distance values are computed for every color region of the query based on (1) the distance between the colors and (2) the distance between the trees. The distance between the trees is computed by comparing the query root with the image root as the first step. Each child of the query node is then iteratively compared against the closest child of the matching image node.

Another image querying technique has been proposed [Jacobs *et al* 1995]. In this system, the user query and the database images are decomposed using standard two-dimensional Haar wavelet decomposition — which involves a one-dimensional decomposition on each row of the image, followed by a one-dimensional decomposition on each column of the result. With the observation that the resulting wavelet coefficients of decomposition can be truncated without losing much discriminating power, only the significant coefficients for each color channel (such as Y, I, and Q) are kept. In order to speed the search and reduce the storage, each truncated coefficient is then quantized to two levels representing the presence or the absence of the features. As a result, for the query and the images, the overall average color as well as the indices and signs of the significant coefficients in each color channel are stored. Having the expectation that a vast majority of database images may not match the query well at all, the similarity score is computed by counting the number of matching coefficients in query and image. Like the region-based method in QBIC, this image querying technique cannot handle subimage queries.

Recently, Chen *et al* presented a formal framework for designing search algorithms which can identify the desired images by spatial distribution of color [Chen *et al* 1997]. The framework is based on a multiscale representation of both the image data and the associated parameter space that must be searched. To process the query, a branch-and-bound algorithm and a multiscale distance function are used. With the multiscale representation, both the query and the images are decomposed into a pyramid of feature vectors. Each level of the multiscale tree represents a scale, and each node at a particular scale contains a feature vector for the corresponding region of the images. At the coarsest scale, the entire image is represented as a single node. For nodes farther away from the root, a more spatially localized region of the image is represented. The distance value at a particular scale is computed by adding the distances between the corresponding feature vectors of the nodes of the query and the image trees at that scale.

Definition 2.1 To conduct the branch-and-bound search for retrieving the top u images from the collection:

1. Check all the images in the database at the coarse scale in the first iteration, and compute the distance value for each image.
2. Sort all images in non-descending order of distance value.
3. Repeat until all top u images have reached the finest scale:
   (a) For each of the u images with smallest distance values,
      i. if the image has not reached the finest scale, extend one level/scale;
      ii. update the distance value of the extended image with the sum of the distances between the corresponding feature vectors of the query and the image at the extended scale.
   (b) Sort all images in non-descending order of distance value. ■

Notice that at each iteration of the branch-and-bound search, the feature vectors used in the computation of distance values may be at a different level/scale and may be for images different from those in the previous iteration. Unless the feature vectors of all levels and for all images can be stored in memory, jumping back and forth in the data file so as to get the necessary feature vectors for computation seems unavoidable. The frequent jumping among different images and scales make it hard to optimize file organization and buffer management, and may generate a large number of I/Os.

Example 2.1 In many database applications, it is not unusual to retrieve the desired images from a collection of thousands of images. For simplicity, in this example, we try to find the top two images from a collection of five images using the branch-and-bound search.

Let Scale 3 be the coarsest scale, and let Scale 0 be the finest scale. For the scales in between, the index decreases as the scale becomes finer. In the first iteration of the branch-and-bound search, all five images are checked at the coarse scale (Scale 3), and a distance value is computed for each image. These images are then sorted in non-descending order of distance value, and the images with the two smallest values are the potential top two images.

	Image 1	Image 2	Image 3	Image 4	Image 5
Iteration 1	Scale 3 val: 25	Scale 3 val: 5	Scale 3 val: 65	Scale 3 val: 80	Scale 3 val: 10
	⇒ potential top two images are Images 2 and 5				

In subsequent iterations, the potential top two images are extended one level/scale and their distance values are updated; the images are then sorted and a new set of potential top two images is obtained.

	Image 1	Image 2	Image 3	Image 4	Image 5
Iteration 2		Scale 2			Scale 2
		val: 30			val: 15
	⇒ potential top two images are Images 5 and 1				
Iteration 3	Scale 2				Scale 1
	val: 35				val: 20
	⇒ potential top two images are Images 5 and 2				
Iteration 4		Scale 1			Scale 0
		val: 32			val: 21
	⇒ potential top two images are Images 5 and 2				
Iteration 5		Scale 0			
		val: 33			
		⇒ top two images are Images 5 and 2			

■

To aim for efficient retrieval, feature vectors at each level of the multiscale systems are extracted and stored prior to runtime. Since the system performance is expected to be affected by the number of levels, we need to choose the number of levels carefully through analyses and experiments. However, detailed analytical and experimental results for the determination of a suitable number of levels are seldom provided in depth.

2.5 Subimage Query Processing

Many of the current IDBMSs support only whole-image matching, but not subimage matching. However, in many situations, users are often interested in local image contents. Due to the poor human memory capability for retaining a fine granularity of spatial information of color, users, in other situations, cannot recall all details of images they have seen before. With whole-image matching, users need to come up with the color distribution on the remaining portion of the images which they may not know or care about. Hence, subimage matching is needed.

To deal with subimage queries of arbitrary size, many template-based algorithms have been proposed. These include the use of traditional template matching techniques based on Euclidean distance for searching for the occurrence of a textured pattern inside each image in the database [Stone and Li 1995], and the use of classified template matching techniques in which templates are classified according to texture and edge features [Tao and Dickinson 1996]. One problem with template-based algorithms is that they require a huge amount of computation due to the large number of positions to be compared. Another problem is that image contents are usually not precomputed or stored; as a result, query processing becomes very slow. Hence, other algorithms for handling subimage queries efficiently and effectively are needed.

The RITAS system [Tao *et al* 1997] is a segmentation-based approach for retrieving textured images containing a pattern similar to the template. Here, each image in the database is divided using a quadtree segmentation technique. A quadtree is built by repeated low-pass filtering and down-sampling. At the coarsest level, feature clustering is performed. Then a boundary refinement procedure is designed to improve the boundary each time when moving down from a higher level. After the segmentation process, for each of the segments, mean and standard deviation of texture energy measures are computed and stored in n-dimensional feature vectors. An n-dimensional spatial relationship vector

is also created for each segment. During the query processing step, the query is segmented in a similar manner. The distance between the query and the image is computed by adding two measures, namely Relational Distance (measuring the difference in spatial locations) and Sum of Minimum Distance. For each query segment, a weighted Euclidean distance between the query segment and its most closely matched image segment is computed. The Sum of Minimum Distance can then be calculated by summing all weighted Euclidean distances. Notice that the system is built specifically for handling textured images. The efficiency and the effectiveness of this approach in handling color and spatial similarity are unknown.

White and Jain presented a framework for developing engines that support subimage matching [White and Jain 1997]. The image representation used in their ImageGREP Engine is a set of binary images stored as bitmaps. The set of binary images is computed by running a bank of 166 color classifiers on the input image. Each color classifier maps an input image into a single binary image by determining whether an input image region corresponding to one bit in the output bitmap contains more than the threshold number of pixels quantized to that color. With this representation, each region is denoted by one of the Present or Absent states, and images can be searched using bitwise operations during the runtime. To take account of subimage translation, all allowable translations are enumerated and stored. In general, the query processing time increases with the number of allowable translations.

As noticed, not many IDBMSs can handle arbitrary-size subimage queries based on color and spatial similarity. For the systems that can deal with subimage queries of arbitrary size, multiscale matching is rarely used.

2.6 Summary

Several research projects have been carried out to aim for efficient and effective content-based retrieval systems. To achieve the goals, one of the important objectives is to provide efficient indexing methods. Many multi-dimensional indexing structures have been designed to help improving efficiency for feature vectors that capture image contents. However, for high dimensional vectors, the "dimensionality curse" problem arises. To tackle the problem, multi-level filtering approaches (which perform preliminary matching on the low dimensional abstracted vectors and detailed matching on the original high dimensional vectors of potential candidates) have been proposed, and information preserving transformations (which compress or reduce the dimensionality of the vector space) can be applied.

The second objective is to develop expressive query languages. With them, efficient querying from image databases for applications with dense and sparse image spaces can be accommodated.

The support of efficient and effective query processing and optimization marks the third important objective. With the popularity of similarity matching on color and spatial information, many IDBMSs store the information in local color histograms. To process user queries, some IDBMSs use fixed grid segmentation approaches. For further improvement on the efficiency and the effectiveness of CBR, multiscale systems have been proposed. However, detailed analytical and experimental results on the determination of the suitable number of levels for these systems are seldom reported, and comparisons of different strategies for searching multiple scales are also rare. Moreover, in many situations, users are interested in, or can remember, only local image contents; therefore, subimage query processing is needed. Unfortunately, not many IDBMSs can handle arbitrary-size subimage queries based on color and spatial similarity. For the systems that can deal with subimage queries of arbitrary size, multiscale matching is rarely used.

Chapter 3

Padding and Reduction Algorithms

Many IDBMSs support whole-image queries, which specify the contents of the whole images to be retrieved. However, users may only remember or care about certain parts of the images. To answer queries of this kind, some systems segment an image into several blocks, each of which has an associated color histogram. One problem with this arrangement is that subimage queries may be of arbitrary size, and not necessarily an integral multiple of the chosen block size. To handle the complication of arbitrary size, some systems use template-based matching algorithms. A key problem with those algorithms is that a huge amount of computation is needed, because of the large number of positions to be compared. As such, we propose two algorithms, called Padding and Reduction, for dealing with subimage queries of arbitrary size.

3.1 Lower Bound to Histogram Distance

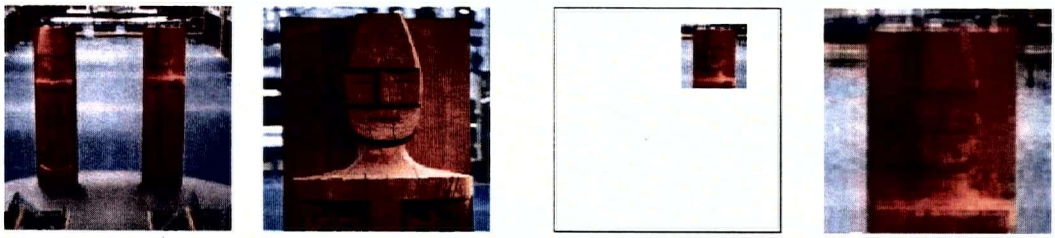
In many IDBMSs, color is extracted automatically and stored in n-dimensional color histograms. Once the color histograms are created for the images in the database, there are a variety of ways to compute the similarity between the feature vectors of the whole-image query and the image. One popular way is based on the Euclidean distance between the color histograms:

$$(\text{Query}_{\text{Histogram}} - \text{Image}_{\text{Histogram}})^T (\text{Query}_{\text{Histogram}} - \text{Image}_{\text{Histogram}})$$

This measure can also be used in computing similarity between the feature vectors of the subimage query and the image subregion even if the query is not of the same size as the image subregion. However, comparing these two vectors may seem unfair as one of the vectors may contain more pixels than another. Due to the quadratic nature of the above Euclidean measure, the excessive pixels in one vector may dramatically influence the difference in the amount of a given color, and hence the resulting distance. For example, given that a query and three image subregions are represented by 3-dimensional vectors Q=(2, 4, 52)^T, I_1=(16, 18, 66)^T, I_2=(44, 4, 52)^T, and I_3=(27, 29, 44)^T respectively, and that the color distribution of the query is the same as a portion of each of the first two (but not the third) image subregions, then the third subregion is a "poorer" match than the first two. However, using the above measure, the distance between Q and I_3 is 1314, which lies in between 588 (the distance between Q and I_1) and 1764 (the distance between Q and I_2). Thus, with this measure of histogram distance, it is not easy to provide a satisfactory discriminating power.

Alternatively, we can use normalized histograms, in which the percentage (instead of the pixel counts as in the standard histograms) of each color is stored; however, if the user is confident in the query size when specifying his query, the use of normalized histograms may not be helpful. For example, given the Query and the two Images shown in

Figure 3.1, Image 1 is the better match. Unfortunately, using normalized histograms with the above Euclidean measure has an effect of scaling up the Query [1]. As a result, Image 1 can no longer match the Query perfectly; instead, Image 2 becomes a better match when operated with normalized histograms. In other words, to maintain the accuracy, the size of a user query is restricted to the size of the image or image subregion. However, our goal is to deal with subimage queries of arbitrary size.



(a) Image 1 (b) Image 2 (c) Query (d) "Scaled" Query

Figure 3.1: The Use of Normalized Histograms

To avoid the problem caused by the size differential between the query and the image, the histogram distance (D_H) is computed between the color histogram Q of the subimage query and the color histogram I of the equal-sized image subregion as $D_H = (Q - I)^T(Q - I)$.

Given a subimage query consisting of $s \times s$ pixels, each image of size $S \times S$ pixels (where $S \geq s$) contains $\frac{S(S+1)(2S+1)}{6}$ potential subregions with size ranging from 1×1 pixel to $S \times S$ pixels. Pre-extraction of color features for all these subregions becomes impractical. Despite that, feature vectors can still be precomputed and stored for some, but not all, subregions of the images in the database. As a result, given a subimage query of arbitrary size, the image subregion represented by the precomputed feature vector may not necessarily be of the same size as the query. Without loss of generality, we assume that:

- the subimage query is square and consists of v pixels, and

[1] The scale-up occurs unless the Query is of the same size as the Images.

- the image subregion consists of w pixels (where $v \leq w$).

Then, instead of computing the exact histogram distance (D_H), we estimate its best-case lower bound ($\widehat{D_H}$). The idea behind the estimation is that we "modify" either the query or the image subregion so that both the query and the image subregion contain the same number of pixels after the "modification" process. The estimated lower bound $\widehat{D_H}$ can then be established by computing the best-case similarity between the feature vectors of the "modified" query and the image subregion, or between the feature vectors of the query and the "modified" image subregion.

Two approaches for estimating the histogram distance between the subimage query and the image subregion are proposed:

1. Padding Approach

   We enlarge the subimage query by padding $w-v$ "desired" pixels to it so that the resulting padded query is of the same size as the image subregion. In order to minimize the distance measure, the "desired" pixels are chosen from the image subregion.

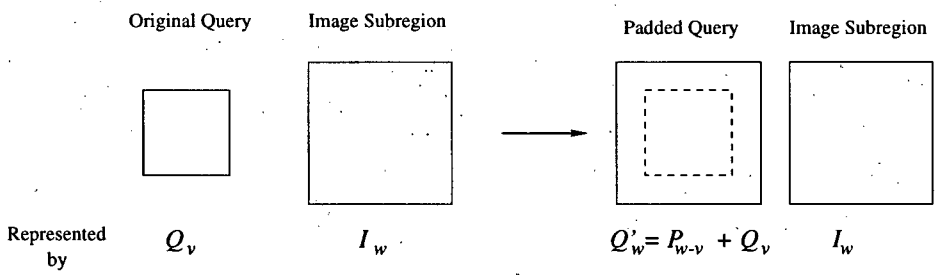


Figure 3.2: Padding Approach

Definition 3.1 Let I, P, Q and Q' represent the histograms of the image subregion, the padded area, the original subimage query, and the resulting padded query respectively, and let the subscripts w, v and $w-v$ indicate the number of pixels represented in the histograms. The goal of the Padding Approach is to find an appropriate assignment to the optimization variable P so that $\widehat{D_H}$ is minimized and

the vector inequality $P_{w-v} \leq I_w$ is met. More precisely, we want to get the optimal P (denoted by P^*) satisfying the condition:

$$\min_{P_{w-v}} (\overbrace{P_{w-v} + Q_v}^{Q'_w} - I_w)^T (\overbrace{P_{w-v} + Q_v}^{Q'_w} - I_w) \quad (3.1)$$

$$\text{such that } P_{w-v} \leq I_w$$

As a result, the estimated best-case lower bound $\widehat{D_H} = (P^*_{w-v} + Q_v - I_w)^T (P^*_{w-v} + Q_v - I_w)$. ■

2. Reduction Approach

We reduce the precomputed image subregion by choosing v "desired" pixels from it so that the resulting reduced image subregion is of the same size as the subimage query. In other words, the $w - v$ "not so desired" pixels are removed.

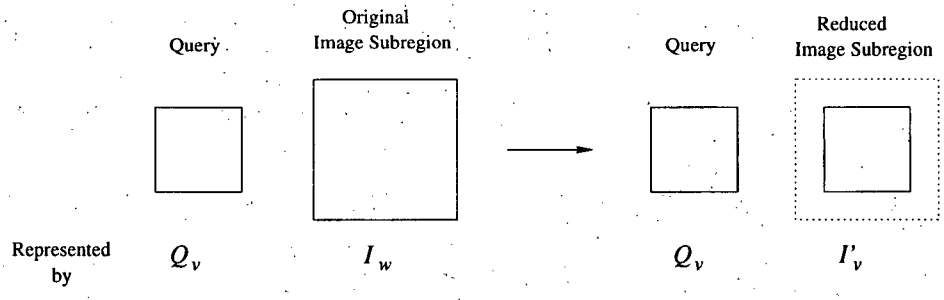


Figure 3.3: Reduction Approach

Definition 3.2 Let I, I' and Q represent the histograms of the original precomputed image subregion, the resulting reduced image subregion, and the subimage query respectively, and let the subscripts w and v indicate the number of pixels represented in the histograms. The goal of the Reduction Approach is to find an appropriate assignment to the optimization variable I' so that $\widehat{D_H}$ is minimized and the vector inequality $I'_v \leq I_w$ is met. More precisely, we want to get the optimal I' (denoted

by I'^*) satisfying the condition:

$$\min_{I'_v} (Q_v - I'_v)^T (Q_v - I'_v) \qquad (3.2)$$
$$\text{such that } I'_v \leq I_w$$

As a result, the estimated best-case lower bound $\widehat{D_H} = (Q_v - I'^*_v)^T (Q_v - I'^*_v)$. ■

3.2 Development of Padding and Reduction Algorithms

Given that the subimage query and the image subregion are represented by n-dimensional color histograms (n-dimensional vectors Q and I), mathematically, the two proposed approaches can be restated as:

- Padding Approach

$$\begin{aligned}
&\text{Given } \sum_{j=1}^{n} Q_j = v \text{ and } \sum_{j=1}^{n} I_j = w \geq v, \\
&\text{find an optimal vector } P^* \text{ to} \\
\textit{objective function} \quad &\text{minimize } (P+Q-I)^T (P+Q-I) \qquad (3.3) \\
\textit{inequality constraint} \quad &\text{subject to } 0 \leq P_j \leq I_j \text{ for } 1 \leq j \leq n \\
\textit{summation constraint} \quad &\text{and } \sum_{j=1}^{n} P_j = w - v \\
\textit{domain constraint} \quad &\text{and } I, P, Q \text{ integer vectors}
\end{aligned}$$

- Reduction Approach

$$\begin{aligned}
&\text{Given } \sum_{j=1}^{n} Q_j = v \text{ and } \sum_{j=1}^{n} I_j = w \geq v, \\
&\text{find an optimal vector } I'^* \text{ to} \\
\textit{objective function} \quad &\text{minimize } (Q-I')^T (Q-I') \qquad (3.4) \\
\textit{inequality constraint} \quad &\text{subject to } 0 \leq I'_j \leq I_j \text{ for } 1 \leq j \leq n \\
\textit{summation constraint} \quad &\text{and } \sum_{j=1}^{n} I'_j = v \\
\textit{domain constraint} \quad &\text{and } I, I', Q \text{ integer vectors}
\end{aligned}$$

3.2.1 Generate-and-Test

A naïve method to solve for the optimal vector P^* or I'^* is to exhaustively find all the possible vectors in which the constraints (inequality, summation, and domain constraints) are satisfied, and then select the vector that gives the minimal value to the objective function. More precisely, for the Padding Approach, all feasible vectors for P are systematically generated by enumerating the value for each entry P_j in the n-dimensional vector. The generated vectors are then tested for minimality, and the one that gives the minimal Euclidean distance is returned:

PADDING-GENERATE&TEST

1 $\widehat{D_H} \leftarrow +\infty$
2 **for** $P_1 \leftarrow 0$ **to** $\min(I_1, w - v)$ **do**
3 **for** $P_2 \leftarrow 0$ **to** $\min(I_2, w - v - P_1)$ **do**
4 **for** $P_3 \leftarrow 0$ **to** $\min(I_3, w - v - P_1 - P_2)$ **do**
5 $\vdots$
6 **for** $P_{n-1} \leftarrow 0$ **to** $\min(I_{n-1}, w - v - \sum_{j=1}^{n-2} P_j)$ **do**
7 $P_n \leftarrow w - v - \sum_{j=1}^{n-1} P_j$
8 **if** $P_n \leq I_n$
9 **then** distance $\leftarrow (P + Q - I)^T (P + Q - I)$
10 **if** distance $< \widehat{D_H}$
11 **then** $\widehat{D_H} \leftarrow$ distance
12 $P^* \leftarrow (P_1, P_2, P_3, \cdots, P_{n-1}, P_n)^T$

Similarly, for the Reduction Approach, all feasible vectors for I'^* are systematically generated by enumerating the value for each entry I'_j in the n-dimensional vector. The generated vectors are then tested for minimality, and the one that gives the minimal

Euclidean distance is reported:

REDUCTION-GENERATE&TEST

1 $\widehat{D_H} \leftarrow +\infty$

2 **for** $I'_1 \leftarrow 0$ **to** $\min(I_1, v)$ **do**

3 **for** $I'_2 \leftarrow 0$ **to** $\min(I_2, v - I'_1)$ **do**

4 **for** $I'_3 \leftarrow 0$ **to** $\min(I_3, v - I'_1 - I'_2)$ **do**

5 $\vdots$

6 **for** $I'_{n-1} \leftarrow 0$ **to** $\min(I_{n-1}, v - \sum_{j=1}^{n-2} I'_j)$ **do**

7 $I'_n \leftarrow v - \sum_{j=1}^{n-1} I'_j$

8 **if** $I'_n \leq I_n$

9 **then** distance $\leftarrow (Q - I')^T (Q - I')$

10 **if** distance $< \widehat{D_H}$

11 **then** $\widehat{D_H} \leftarrow$ distance

12 $I'^* \leftarrow (I'_1, I'_2, I'_3, \cdots, I'_{n-1}, I'_n)^T$

Like many generate-and-test applications, this naïve method of solving for P^* and I'^* is unpalatable in the sense that the execution time is expected to be very long.

3.2.2 Quadratic Programming with Integer Programming

With the observation that the tasks of finding the vector P^* in Problem (3.3) and the vector I'^* in Problem (3.4) are instances of quadratic programming (QP) problems, existing mathematical software packages can be used. Examples of these software packages include MATLAB (MATrix LABoratory) [Sigmon 1992] and LINDO (Linear INteractive Discrete Optimizer) [Schrage 1986]. In order to use the software packages, the minimization problems are usually required to be converted into the forms:

- Padding Approach

$$\min_{P} \frac{1}{2}P^T(2\mathbf{I})P + [2(Q-I)]^T P \tag{3.5}$$

$$\text{s.t.} \left(\begin{array}{c} \begin{array}{cccc} -1 & -1 & \cdots & -1 \end{array} \\ \hline \begin{array}{cccc} 1 & 1 & \cdots & 1 \end{array} \\ \hline -\mathbf{I} \\ \hline \mathbf{I} \end{array}\right) P \leq \left(\begin{array}{c} v-w \\ \hline w-v \\ \hline 0 \\ \hline I \end{array}\right)$$

where $\mathbf{I}$ is the $n \times n$ identity matrix

- Reduction Approach

$$\min_{I'} \frac{1}{2}(I')^T(2\mathbf{I})(I') + [-2Q]^T(I') \tag{3.6}$$

$$\text{s.t.} \left(\begin{array}{c} \begin{array}{cccc} -1 & -1 & \cdots & -1 \end{array} \\ \hline \begin{array}{cccc} 1 & 1 & \cdots & 1 \end{array} \\ \hline -\mathbf{I} \\ \hline \mathbf{I} \end{array}\right) I' \leq \left(\begin{array}{c} -v \\ \hline v \\ \hline 0 \\ \hline I \end{array}\right)$$

where $\mathbf{I}$ is the $n \times n$ identity matrix

One problem with the software packages is that the computation of optimal vectors is usually done in the domain of real numbers, not the domain of integers. The domain

problem coupled with the roundoff error may lead to the unreliability of some output vectors. For example, given I = (1430, 3257, 6133)^T and Q = (1429, 0, 2)^T, the software package outputs (1430, 3257, 4702)^T as the answering vector (with value for objective function = 4084082) for the Padding Approach, but the expected optimal vector P^* is (1, 3257, 6131)^T with the corresponding optimal value of 0. As another example, given I = (2, 7, 5)^T and Q = (5, 1, 1)^T, an expected optimal integer vector I'^* is (2, 3, 2)^T, but the software package returns the real vector (2, 2.5, 2.5)^T to represent the reduced image subregion. Hence, we want the method to handle not only QP, but also integer programming (IP). With IP, the domain constraints — I, P, Q integer vectors and I, I', Q integer vectors — can be specified. Unfortunately, the IP function is rarely supported in conjunction with QP; in many software packages, the IP function is not supported at all.

3.2.3 Algorithms PAD and RED

Since the above methods may sometimes be unsound or time consuming, efficient and effective methods for estimating the best-case lower bound to the histogram distance are needed. It is well known that for a vector x, $x^T x$ can be written as $\sum(x_j)^2$. So, given $\sum_{j=1}^{n} Q_j = v$ and $\sum_{j=1}^{n} I_j = w \geq v$, Problems (3.3) and (3.4) can be rewritten as:

$$\begin{aligned} \widehat{D_H} = \min_{P} & \sum_{j=1}^{n} (P_j - \alpha_j)^2 \\ & \text{subject to } 0 \leq P_j \leq I_j \text{ for } 1 \leq j \leq n \\ & \text{and } \sum_{j=1}^{n} P_j = w - v \\ & \text{and } I, P, \alpha \text{ integer vectors} \end{aligned} \tag{3.7}$$

where $\alpha = I - Q$; and

$$\widehat{D_H} = \min_{I'} \sum_{j=1}^{n} (I'_j - Q_j)^2 \qquad (3.8)$$
$$\text{subject to } 0 \le I'_j \le I_j \text{ for } 1 \le j \le n$$
$$\text{and } \sum_{j=1}^{n} I'_j = v$$
$$\text{and } I, I', Q \text{ integer vectors}$$

respectively.

With these representations, each of the two objective functions is in the form of the sum of squares of the difference terms:

$$\sum (P_j - \alpha_j)^2 \quad \text{or} \quad \sum (I'_j - Q_j)^2$$

In order to minimize the sum, we need to minimize the difference terms. In which order should the terms be minimized ? Due to the quadratic nature of the squares of the difference terms, we note, on a close examination of the representations, that deducting 1 off a large difference term is more effective in minimizing the sum than deducting 1 off a small difference term:

$$\text{If integers } a > b \ge 1, \text{ then } (a-1)^2 + b^2 \; < \; a^2 + (b-1)^2$$

It is also clear that for two difference terms having the same value c, subtracting an integer d from each of the these difference terms is more effective in minimizing the sum than subtracting $2d$ from only one of these two equal-valued terms:

$$\text{If integers } c > 2d \text{ and } d \ge 1, \text{ then } (c-d)^2 + (c-d)^2 < (c-2d)^2 + c^2$$

Hence, for the Padding Approach, we start with $P_j = 0$ for all j, and each difference term $(P_j - \alpha_j)$ becomes $-\alpha_j$. These terms are then rearranged in non-ascending order of α_j (in other words, non-descending order of $-\alpha_j$) and result in:

$$\{(P_{(1)} - \alpha_{(1)}), (P_{(2)} - \alpha_{(2)}), \cdots, (P_{(n)} - \alpha_{(n)})\} \text{ where } \alpha_{(1)} \ge \alpha_{(2)} \ge \cdots \ge \alpha_{(n)}$$

After the rearrangement, we try to lower the difference in the first term $(P_{(1)} - \alpha_{(1)})$ by adding the value $\alpha_{(1)} - \alpha_{(2)}$ to $P_{(1)}$ [2] so that the first term has the same value as the second difference term $(P_{(2)} - \alpha_{(2)})$. We then try to reduce the values of these first two difference terms by increasing the values of $P_{(1)}$ and $P_{(2)}$ in round-robin fashion [3] so as to make them have the same difference as the third term $(P_{(3)} - \alpha_{(3)})$. We keep devaluing the first k difference terms so that they have the same value as the $(k+1)$-th difference term through increases of the values of $P_{(1 \le j \le k)}$ [4]. This process is repeated until the summation constraint $\sum_{j=1}^{n} P_j = w - v$ is satisfied. At any cycle, the value of $P_{(j)}$ is constrained by the inequality $P_{(j)} \le I_{(j)}$ and will not be increased beyond its allowable maximum $I_{(j)}$.

Algorithm 3.1 (Algorithm PAD)

1 $\forall j,\ \alpha_j \leftarrow I_j - Q_j$

2 $\forall j,\ P_j \leftarrow 0$

3 $\{\ (P_{(j)} - \alpha_{(j)})\ \} \leftarrow$ sort the $(P_j - \alpha_j)$ terms in non-ascending order of α_j

4 $k \leftarrow 1$

5 **while** $k < n$ **and** $\sum_{j=1}^{k} P_{(j)} < w - v$ **do**

6 **loop** for at most $\alpha_{(k)} - \alpha_{(k+1)}$ cycles

7 **for each** $P_{(1 \le t \le k)}$ **do**

8 **if** $P_{(t)} < I_{(t)}$ **then** $P_{(t)} \leftarrow P_{(t)} + 1$

9 **if** $\sum_{j=1}^{k} P_{(j)} = w - v$ **then return** P

10 $k \leftarrow k + 1$

11 **if** $\sum_{j=1}^{n} P_{(j)} < w - v$

12 **then loop**

13 **for each** $P_{(1 \le t \le n)}$ **do**

[2] If $P_{(1)}$ reaches its allowable maximum $I_{(1)}$, the value of $P_{(1)}$ will not be increased in subsequent cycles. In such a case, the difference in the first term may not be the same as the difference in the second term.

[3] Again, if $P_{(j)} \in \{P_{(1)}, P_{(2)}\}$ reaches its allowable maximum $I_{(j)}$, the value of $P_{(j)}$ will not be increased in subsequent cycles.

[4] Similarly, if $P_{(j)} \in \{P_{(1)}, \cdots, P_{(k)}\}$ reaches its allowable maximum $I_{(j)}$, the value of $P_{(j)}$ will not be increased in subsequent cycles.

14 **if** $P_{(t)} < I_{(t)}$ **then** $P_{(t)} \leftarrow P_{(t)} + 1$

15 **if** $\sum_{j=1}^{n} P_{(j)} = w - v$ **then return** P

Example 3.1 Given integer vectors $I = \begin{pmatrix} 12 \\ 3 \\ 10 \end{pmatrix}$ and $Q = \begin{pmatrix} 2 \\ 7 \\ 1 \end{pmatrix}$; then, $\alpha = I - Q = (\ 10,\ -4,\ 9\ )^T$. We want to find an appropriate assignment to integer vector $(\ P_1,\ P_2,\ P_3\ )^T$ so that the objective function $(P_1 - 10)^2 + (P_2 + 4)^2 + (P_3 - 9)^2$ is minimized and the constraints $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \leq \begin{pmatrix} P_1 \\ P_2 \\ P_3 \end{pmatrix} \leq \begin{pmatrix} 12 \\ 3 \\ 10 \end{pmatrix}$ and $\sum_{j=1}^{3} P_j = 25 - 10 = 15$ are satisfied.

We start with $P_j = 0$ for all j and rearrange all the difference terms. After the rearrangement, we try to lower the difference in the first term ($P_{(1)} - 10 = -10$) by adding 1 to $P_{(1)}$ so that the first two terms have same difference ($= -9$). Then, we try to reduce the values of these two difference terms by increasing the values of $P_{(1)}$ and $P_{(2)}$ in round-robin fashion so as to bring them closer to the value of the third difference term ($= 4$); we stop when the constraint $\sum_{j=1}^{3} P_j = 15$ is satisfied.

		$P_{(1)} - \alpha_{(1)}$ $= P_1 - \alpha_1$	$P_{(2)} - \alpha_{(2)}$ $= P_3 - \alpha_3$	$P_{(3)} - \alpha_{(3)}$ $= P_2 - \alpha_2$	$\sum P_j$
	Cycle 0	$0 - 10 = -10$	$0 - 9 = -9$	$0 - (-4) = 4$	0
$k = 1$:	Cycle 1	$1 - 10 = -9$			1
$k = 2$:	Cycle 2	$2 - 10 = -8$	$1 - 9 = -8$		3
	Cycle 3	$3 - 10 = -7$	$2 - 9 = -7$		5
	⋮	⋮	⋮		⋮
	Cycle 8	$8 - 10 = -2$	$7 - 9 = -2$		15

Algorithm PAD returns $P^* = \begin{pmatrix} 8 \\ 0 \\ 7 \end{pmatrix}$ with $\sum(P_j^* - \alpha_j)^2 = 24$ ■

Similarly, for the Reduction Approach, we start with $I'_j = 0$ for all j, and each difference term $(I'_j - Q_j)$ becomes $-Q_j$. These terms are then rearranged in non-ascending order of Q_j (in other words, non-descending order of $-Q_j$) and result in:

$$\{(I'_{(1)} - Q_{(1)}), (I'_{(2)} - Q_{(2)}), \cdots, (I'_{(n)} - Q_{(n)})\} \text{ where } Q_{(1)} \geq Q_{(2)} \geq \cdots \geq Q_{(n)}$$

After the rearrangement, we try to lower the difference in the first term $(I'_{(1)} - Q_{(1)})$ by adding the value $Q_{(1)} - Q_{(2)}$ to $I'_{(1)}$ [5] so that the first term has the same value as the second difference term $(I'_{(2)} - Q_{(2)})$. We then try to reduce the values of these first two difference terms by increasing the values of $I'_{(1)}$ and $I'_{(2)}$ in round-robin fashion [6] so as to make them have the same difference as the third term $(I'_{(3)} - Q_{(3)})$. We keep devaluing the first k difference terms so that they have the same value as the $(k+1)$-th difference term through increases of the values of $I'_{(1 \leq j \leq k)}$ [7]. This process is repeated until the summation constraint $\sum_{j=1}^{n} I'_j = v$ is satisfied. At any cycle, the value of $I'_{(j)}$ is constrained by the inequality $I'_{(j)} \leq I_{(j)}$ and will not be increased beyond its allowable maximum $I_{(j)}$.

Algorithm 3.2 (Algorithm RED)

1 $\forall j,\ I'_j \leftarrow 0$

2 $\{\ (I'_{(j)} - Q_{(j)})\ \} \leftarrow$ sort the $(I'_j - Q_j)$ terms in non-ascending order of Q_j

3 $k \leftarrow 1$

4 **while** $k < n$ **and** $\sum_{j=1}^{k} I'_{(j)} < v$ **do**

[5] If $I'_{(1)}$ reaches its allowable maximum $I_{(1)}$, the value of $I'_{(1)}$ will not be increased in subsequent cycles. In such a case, the difference in the first term may not be the same as the difference in the second term.

[6] Again, if $I'_{(j)} \in \{I'_{(1)}, I'_{(2)}\}$ reaches its allowable maximum $I_{(j)}$, the value of $I'_{(j)}$ will not be increased in subsequent cycles.

[7] Similarly, if $I'_{(j)} \in \{I'_{(1)}, \cdots, I'_{(k)}\}$ reaches its allowable maximum $I_{(j)}$, the value of $I'_{(j)}$ will not be increased in subsequent cycles.

5 **loop** for at most $Q_{(k)} - Q_{(k+1)}$ cycles

6 **for each** $I'_{(1 \le t \le k)}$ **do**

7 **if** $I'_{(t)} < I_{(t)}$ **then** $I'_{(t)} \leftarrow I'_{(t)} + 1$

8 **if** $\sum_{j=1}^{k} I'_{(j)} = v$ **then return** I'

9 $k \leftarrow k + 1$

10 **if** $\sum_{j=1}^{n} I'_{(j)} < v$

11 **then loop**

12 **for each** $I'_{(1 \le t \le n)}$ **do**

13 **if** $I'_{(t)} < I_{(t)}$ **then** $I'_{(t)} \leftarrow I'_{(t)} + 1$

14 **if** $\sum_{j=1}^{n} I'_{(j)} = v$ **then return** I'

Example 3.2 Given integer vectors $I = \begin{pmatrix} 12 \\ 3 \\ 10 \end{pmatrix}$ and $Q = \begin{pmatrix} 2 \\ 7 \\ 1 \end{pmatrix}$; we want to find an appropriate assignment to integer vector $(I'_1, I'_2, I'_3)^T$ so that the objective function $(I'_1-2)^2+(I'_2-7)^2+(I'_3-1)^2$ is minimized and the constraints $\begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \le \begin{pmatrix} I'_1 \\ I'_2 \\ I'_3 \end{pmatrix} \le \begin{pmatrix} 12 \\ 3 \\ 10 \end{pmatrix}$ and $\sum_{j=1}^{3} I'_j = 10$ are satisfied.

We start with $I'_j = 0$ for all j and rearrange all the difference terms. After the rearrangement, we try to lower the difference in the first term $(I'_{(1)} - 7 = -7)$ by adding at most 5 to $I'_{(1)}$. Since the value of $I'_{(1)}$ is constrained by the inequality $I'_{(1)} \le 3$, it will not be increased beyond 3. Then, we try to reduce the difference in the second term $(I'_{(2)} - 2 = -2)$ by adding 1 to $I'_{(2)}$ so that the second term has the same value as the third difference term $(= -1)$. We keep devaluing the second and the third difference terms in round-robin fashion until the summation constraint $\sum_{j=1}^{3} I'_j = 10$ is satisfied.

		$I'_{(1)} - Q_{(1)}$ $= I'_2 - Q_2$	$I'_{(2)} - Q_{(2)}$ $= I'_1 - Q_1$	$I'_{(3)} - Q_{(3)}$ $= I'_3 - Q_3$	$\sum I'_j$
	Cycle 0	$0-7=-7$	$0-2=-2$	$0-1=-1$	0
$k=1:$	Cycle 1	$1-7=-6$			1
	Cycle 2	$2-7=-5$			2
	Cycle 3	$3-7=-4$			3
$k=2:$	Cycle 4		$1-2=-1$		4
$k=3:$	Cycle 5		$2-2=0$	$1-1=0$	6
	Cycle 6		$3-2=1$	$2-1=1$	8
	Cycle 7		$4-2=2$	$3-1=2$	10

Algorithm RED returns $I'^* = \begin{pmatrix} 4 \\ 3 \\ 3 \end{pmatrix}$ with $\sum(I'^*_j - Q_j)^2 = 24$ ■

3.3 Analytical Comparison

Having developed two algorithms, namely Algorithm PAD and Algorithm RED, which one produces a better lower bound ?

Theorem 3.1 In the domain of integers, given $\sum_{j=1}^{n} Q_j = v$ and $\sum_{j=1}^{n} I_j = w \geq v$.

- Define $\mathcal{D}_H^{PAD} = (P+Q-I)^T(P+Q-I)$
  such that for $1 \leq j \leq n,\ 0 \leq P_j \leq I_j$
  and $\sum_{j=1}^{n} P_j = w - v$

- Define $\mathcal{D}_H^{RED} = (Q-I')^T(Q-I')$
  such that for $1 \leq j \leq n,\ 0 \leq I'_j \leq I_j$
  and $\sum_{j=1}^{n} I'_j = v$

Then, there is a 1-to-1 correspondence between $\mathcal{D}_H^{PAD}$ and $\mathcal{D}_H^{RED}$.

Proof

[$\Rightarrow$] Let $I' = I - P$. Then, the objective function $(P+Q-I)^T(P+Q-I)$ becomes $(Q-I')^T(Q-I')$.

The inequality constraint $\forall j,\ 0 \leq P_j \leq I_j$ can be rewritten as $\forall j,\ 0 = P_j - P_j \leq I_j - P_j = I'_j \leq I_j$.

The summation constraint $\sum P_j = w - v$ coupled with the equality $\sum P_j = \sum(I_j - I'_j) = \sum I_j - \sum I'_j = w - \sum I'_j$ implies that $\sum I'_j = v$.

[$\Leftarrow$] Let $P = I - I'$. Then, the objective function $(Q-I')^T(Q-I') = (Q-I'+I-I)^T(Q-I'+I-I)$ becomes $(P+Q-I)^T(P+Q-I)$.

The inequality constraint $\forall j,\ 0 \leq I'_j \leq I_j$ can be rewritten as $\forall j,\ 0 = I'_j - I'_j \leq I_j - I'_j = P_j \leq I_j$.

The summation constraint $\sum I'_j = v$ coupled with the equality $\sum I'_j = \sum(I_j - P_j) = \sum I_j - \sum P_j = w - \sum P_j$ implies that $\sum P_j = w - v$.

Therefore, given a vector P, there exists a corresponding vector I' such that $\mathcal{D}_H^{PAD} = \mathcal{D}_H^{RED}$. Similarly, given a vector I', there exists a corresponding vector P such that $\mathcal{D}_H^{PAD} = \mathcal{D}_H^{RED}$. Hence, a bijection between $\mathcal{D}_H^{PAD}$ and $\mathcal{D}_H^{RED}$ exists. ■

Corollary 3.2 Algorithm PAD and Algorithm RED produce the same lower bound to the histogram distance.

Proof Algorithm PAD is an exact algorithm for computing the special case of $\mathcal{D}_H^{PAD}$ — namely the minimal $\mathcal{D}_H^{PAD}$ (denoted by $\min \mathcal{D}_H^{PAD}$) — which estimates the best-case lower bound to the histogram distance D_H. Similarly, Algorithm RED is an exact algorithm for computing the special case of $\mathcal{D}_H^{RED}$ — namely the minimal $\mathcal{D}_H^{RED}$ (denoted by $\min \mathcal{D}_H^{RED}$) — which estimates the best-case lower bound to the histogram distance D_H. Since there is a 1-to-1 correspondence between $\mathcal{D}_H^{PAD}$ and $\mathcal{D}_H^{RED}$, there exists a 1-to-1 correspondence

between their special cases (the minima):

$$\widehat{D_H} = \min \mathcal{D}_H^{PAD} = \min \mathcal{D}_H^{RED}$$

In other words, the values returned by Algorithms PAD and RED are the same estimated best-case lower bound to the histogram distance. ■

Since Algorithms PAD and RED are methods to implement the two proposed approaches, the Padding Approach and the Reduction Approach ideally produce the same best-case lower bound as well.

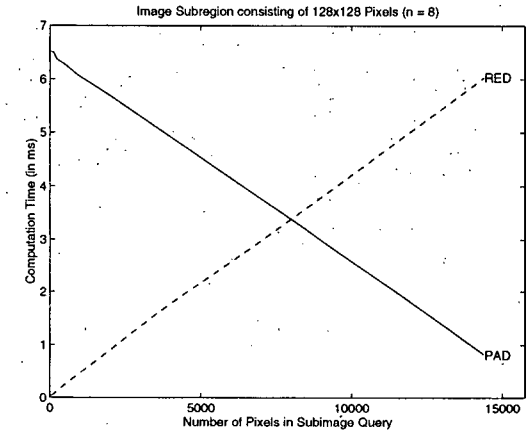
3.4 Experimental Comparison

In terms of accuracy, both Algorithm PAD and Algorithm RED produce the same best-case lower bound. The question is: Which one produces the lower bound faster ? To answer this question, we performed experiments using color histograms of 8, 64, and 512 dimensions. We used 500 images or image subregions of each of the assorted sizes (128×128, 64×64, 32×32, and 16×16 pixels) and 10 subimage queries for each of those sizes. The test queries consisted of 5×5, 15×15, 30×30, 60×60, and 120×120 pixels. The experiments were run on a Sun UltraSPARC-1 workstation. The results (the average computation time per query for each combination of the above mentioned sizes of image subregions and queries) are summarized in the tables and figures on the following pages. In the figures, the time curve for Algorithm PAD is represented by a blue solid line, and the time curve for Algorithm RED is represented by a red dashed line.

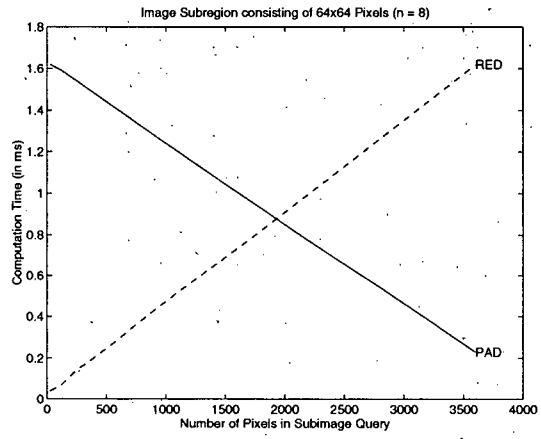
- 8-dimensional color histograms

Image Subregion Size	Query Size	Computation Time	
		Algorithm PAD	Algorithm RED
128×128	5×5	6.52 ± 0.27 ms	0.04 ± 0.00 ms
	11×11	6.50 ± 0.28 ms	0.07 ± 0.00 ms
	15×15	6.37 ± 0.29 ms	0.13 ± 0.01 ms
	23×23	6.26 ± 0.30 ms	0.27 ± 0.01 ms
	30×30	6.08 ± 0.28 ms	0.43 ± 0.01 ms
	45×45	5.68 ± 0.26 ms	0.87 ± 0.05 ms
	60×60	5.07 ± 0.69 ms	1.61 ± 0.21 ms
	91×91	3.37 ± 0.19 ms	3.40 ± 0.87 ms
	120×120	0.82 ± 0.05 ms	6.04 ± 2.52 ms
64×64	5×5	1.62 ± 0.12 ms	0.04 ± 0.00 ms
	11×11	1.59 ± 0.07 ms	0.07 ± 0.01 ms
	15×15	1.55 ± 0.07 ms	0.13 ± 0.02 ms
	23×23	1.43 ± 0.07 ms	0.26 ± 0.05 ms
	30×30	1.28 ± 0.06 ms	0.43 ± 0.12 ms
	45×45	0.89 ± 0.08 ms	0.87 ± 0.49 ms
	60×60	0.23 ± 0.01 ms	1.62 ± 1.08 ms
32×32	5×5	0.42 ± 0.03 ms	0.04 ± 0.01 ms
	11×11	0.40 ± 0.02 ms	0.07 ± 0.02 ms
	15×15	0.35 ± 0.02 ms	0.13 ± 0.06 ms
	23×23	0.24 ± 0.07 ms	0.26 ± 0.18 ms
	30×30	0.08 ± 0.00 ms	0.43 ± 0.31 ms
16×16	5×5	0.12 ± 0.01 ms	0.04 ± 0.01 ms
	11×11	0.08 ± 0.00 ms	0.08 ± 0.04 ms
	15×15	0.04 ± 0.01 ms	0.13 ± 0.10 ms

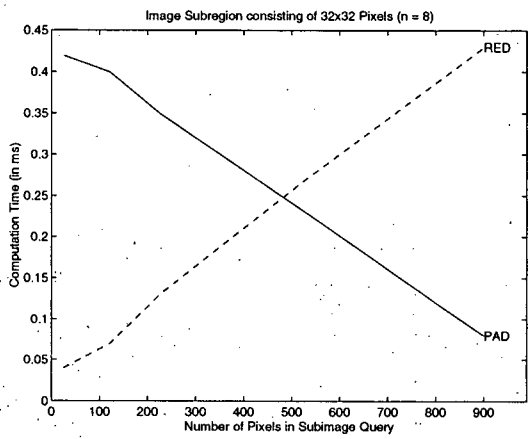
Table 3.1: Computation Time for Algorithms PAD and RED (8-dimensional Histograms)



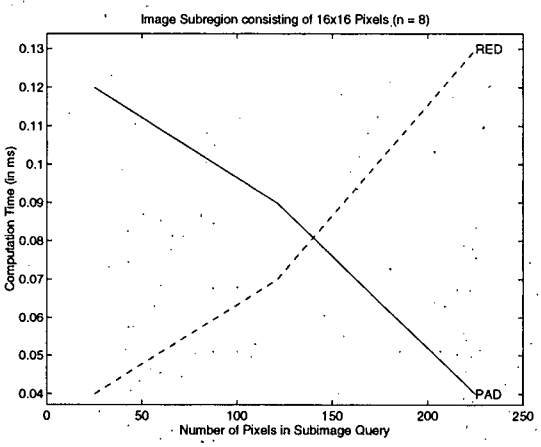
(a) Image Subregion Size = 128×128 Pixels



(b) Image Subregion Size = 64×64 Pixels



(c) Image Subregion Size = 32×32 Pixels



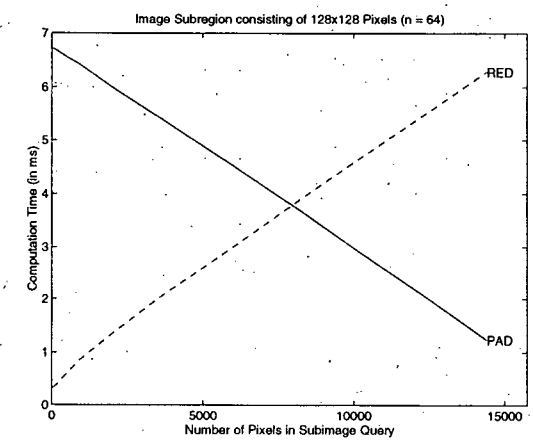
(d) Image Subregion Size = 16×16 Pixels

Figure 3.4: Computation Time for Algorithms PAD and RED (8-dimensional Histograms)

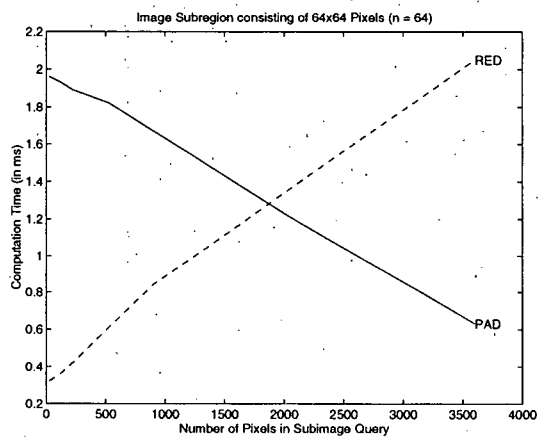
- 64-dimensional color histograms

Image Subregion Size	Query Size	Computation Time	
		Algorithm PAD	Algorithm RED
128×128	5×5	6.71 ± 0.22 ms	0.32 ± 0.05 ms
	11×11	6.69 ± 0.21 ms	0.36 ± 0.02 ms
	15×15	6.65 ± 0.22 ms	0.42 ± 0.03 ms
	23×23	6.54 ± 0.69 ms	0.61 ± 0.10 ms
	30×30	6.42 ± 0.21 ms	0.84 ± 0.22 ms
	45×45	5.98 ± 0.21 ms	1.28 ± 0.64 ms
	60×60	5.41 ± 0.21 ms	2.03 ± 1.38 ms
	91×91	3.77 ± 0.16 ms	3.80 ± 2.40 ms
	120×120	1.23 ± 0.11 ms	6.28 ± 4.58 ms
64×64	5×5	1.96 ± 0.08 ms	0.32 ± 0.01 ms
	11×11	1.93 ± 0.07 ms	0.36 ± 0.04 ms
	15×15	1.89 ± 0.07 ms	0.42 ± 0.11 ms
	23×23	1.82 ± 0.07 ms	0.61 ± 0.32 ms
	30×30	1.67 ± 0.10 ms	0.84 ± 0.66 ms
	45×45	1.29 ± 0.06 ms	1.28 ± 0.75 ms
	60×60	0.63 ± 0.04 ms	2.05 ± 1.59 ms
32×32	5×5	0.75 ± 0.04 ms	0.32 ± 0.03 ms
	11×11	0.71 ± 0.05 ms	0.36 ± 0.13 ms
	15×15	0.66 ± 0.04 ms	0.42 ± 0.34 ms
	23×23	0.60 ± 0.57 ms	0.61 ± 0.25 ms
	30×30	0.43 ± 0.04 ms	0.85 ± 0.47 ms
16×16	5×5	0.42 ± 0.05 ms	0.32 ± 0.06 ms
	11×11	0.38 ± 0.05 ms	0.37 ± 0.24 ms
	15×15	0.33 ± 0.04 ms	0.42 ± 0.29 ms

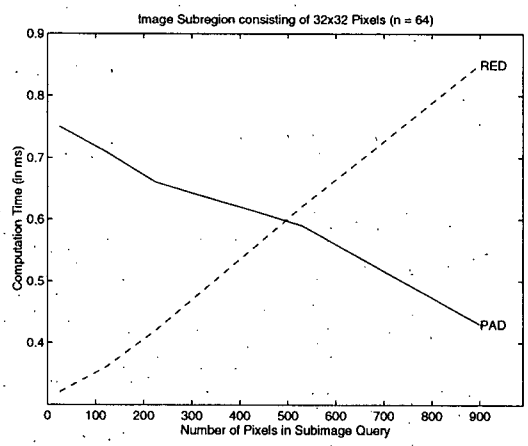
Table 3.2: Computation Time for Algorithms PAD and RED (64-dimensional Histograms)



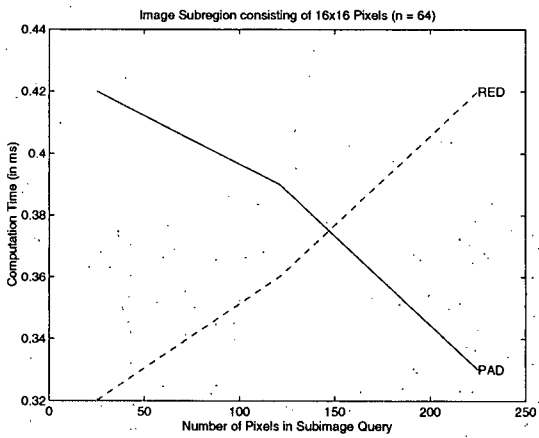
(a) Image Subregion Size = 128×128 Pixels



(b) Image Subregion Size = 64×64 Pixels



(c) Image Subregion Size = 32×32 Pixels



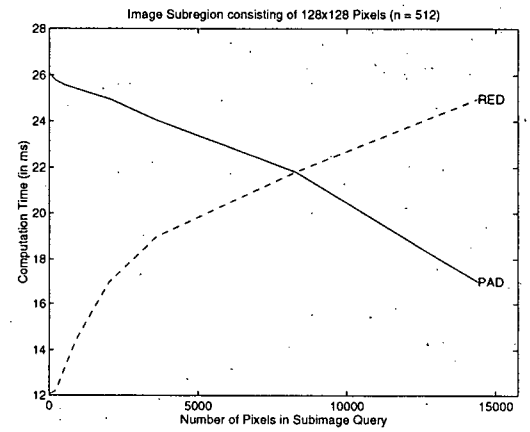
(d) Image Subregion Size = 16×16 Pixels

Figure 3.5: Computation Time for Algorithms PAD and RED (64-dimensional Histograms)

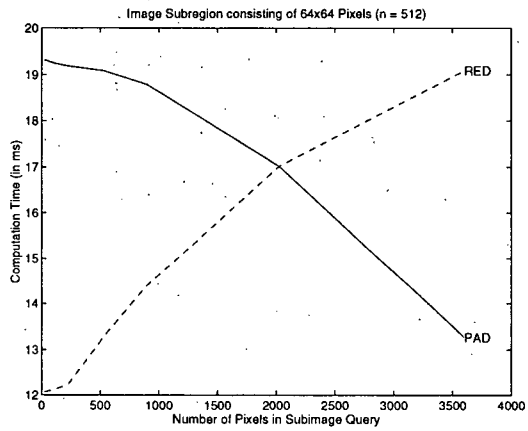
- 512-dimensional color histograms

Image Subregion Size	Query Size	Computation Time	
		Algorithm PAD	Algorithm RED
128×128	5×5	26.05 ± 2.06 ms	12.06 ± 0.16 ms
	11×11	25.91 ± 2.00 ms	12.13 ± 0.64 ms
	15×15	25.78 ± 2.09 ms	12.22 ± 0.38 ms
	23×23	25.58 ± 1.74 ms	13.25 ± 0.85 ms
	30×30	25.43 ± 1.71 ms	15.42 ± 1.44 ms
	45×45	24.98 ± 1.46 ms	16.99 ± 4.56 ms
	60×60	24.06 ± 1.37 ms	18.98 ± 6.56 ms
	91×91	21.82 ± 1.36 ms	21.82 ± 8.49 ms
	120×120	16.99 ± 1.08 ms	24.98 ± 10.34 ms
64×64	5×5	19.32 ± 2.00 ms	12.06 ± 0.60 ms
	11×11	19.24 ± 1.84 ms	12.13 ± 0.37 ms
	15×15	19.19 ± 1.85 ms	12.23 ± 0.66 ms
	23×23	19.09 ± 1.55 ms	13.28 ± 1.71 ms
	30×30	18.79 ± 1.59 ms	15.40 ± 3.57 ms
	45×45	17.02 ± 1.34 ms	17.02 ± 7.24 ms
	60×60	13.27 ± 1.24 ms	19.09 ± 10.82 ms
32×32	5×5	14.50 ± 1.57 ms	12.08 ± 0.34 ms
	11×11	14.45 ± 1.46 ms	12.14 ± 1.08 ms
	15×15	14.23 ± 1.46 ms	12.24 ± 2.97 ms
	23×23	13.28 ± 1.41 ms	13.30 ± 4.43 ms
	30×30	12.14 ± 1.26 ms	15.45 ± 5.51 ms
16×16	5×5	12.27 ± 1.04 ms	12.09 ± 0.38 ms
	11×11	12.16 ± 1.05 ms	12.16 ± 1.70 ms
	15×15	12.09 ± 1.03 ms	12.27 ± 4.00 ms

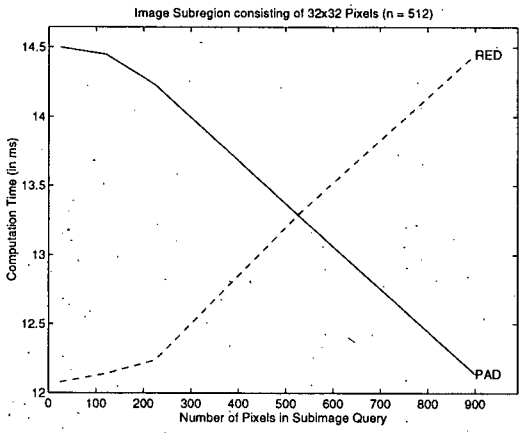
Table 3.3: Computation Time for Algorithms PAD and RED (512-dimensional Histograms)



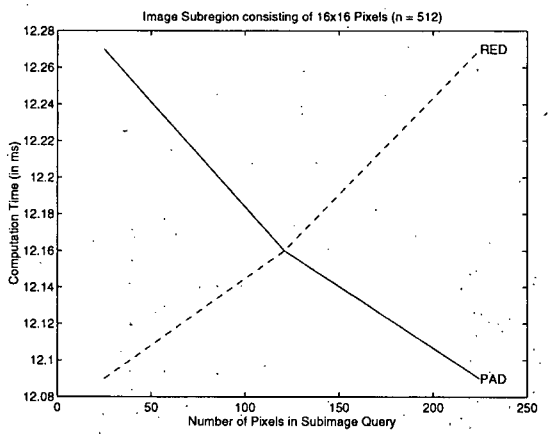
(a) Image Subregion Size $= 128 \times 128$ Pixels



(b) Image Subregion Size $= 64 \times 64$ Pixels



(c) Image Subregion Size $= 32 \times 32$ Pixels



(d) Image Subregion Size $= 16 \times 16$ Pixels

Figure 3.6: Computation Time for Algorithms PAD and RED (512-dimensional Histograms)

Observation 3.1 Computation time is affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required by Algorithms PAD and RED increases.

Explanation In both algorithms, we start with 0 for each of the n entries in the n-dimensional histogram (the vector P^* or I'^*) and keep increasing the value of each entry. Hence, the increase in the dimension of the histogram leads to an increase in the number of entries. Due to the nested loops in the Algorithms for manipulating the n entries in the vectors, the time required increases (though not linearly). ■

Observation 3.2 Computation time is also affected by the number of pixels in the query and the image subregion: Given a fixed-size image subregion, and varying the size of the subimage query, it is observed that as the query size increases, the time required by Algorithm PAD appears to decrease linearly.

Explanation Given an image subregion consisting of w pixels, Algorithm PAD pads $w - v$ "desired" pixels to the subimage query consisting of v pixels. So, as the query increases in size, the number of pixels in the query (= v pixels) also increases; thus, decreasing the number of the pixels needed to be padded (= $w - v$ pixels). ■

Observation 3.3 Given a fixed-size image subregion, and varying the size of the subimage query, it is noticed that as the query size increases, the time taken by Algorithm RED appears to increase linearly.

Explanation Given an image subregion consisting of w pixels, Algorithm RED reduces the subregion by choosing v "desired" pixels. So, as the query increases in size, the number of pixels in the query (= v pixels) also increases; thus, increasing the number of pixels needed to be chosen (= v pixels). ■

Observation 3.4 Given a fixed-size image subregion, and varying the size of the subimage query, it can be viewed from the figures that as the query size increases, the time

curve for Algorithm PAD declines and the time curve for Algorithm RED is rises. The two curves meet at the point representing medium-sized queries. For a small-sized query, the time taken by Algorithm RED is less than that taken by Algorithm PAD; for a large-sized query, the time required by Algorithm RED is more than that required by Algorithm PAD. Lastly, for a medium-sized query, the times needed by the two Algorithms are almost the same as indicated by the intersection of the curves in the figures.

Explanation

- Given an image subregion consisting of w pixels and a small-sized query consisting of v pixels (where $2v < w$), Algorithm RED reduces the image subregion by picking v "desired" pixels, whereas Algorithm PAD pads $w - v > v$ "desired" pixels to the query. Thus, Algorithm RED requires less time.

- Given an image subregion consisting of w pixels and a large-sized query consisting of v pixels (where $v \leq w < 2v$), Algorithm PAD pads $w - v$ "desired" pixels to the query, whereas Algorithm RED reduces the image subregion by choosing $v > w - v$ "desired" pixels. Hence, Algorithm PAD needs less time.

- Given an image subregion consisting of w pixels and a medium-sized query consisting of v pixels (where $v \approx \frac{1}{2}w \Rightarrow 2v \approx w$), Algorithm RED reduces the image subregion by selecting v "desired" pixels and Algorithm PAD pads $w - v \approx v$ "desired" pixels to the query. So, both Algorithms take almost the same amount of computation time. ■

Therefore, if the subimage query is of the same size as the precomputed image subregion, then the exact histogram distance of the form $(Q-I)^T(Q-I)$ can be applied. Otherwise, use Algorithm PAD when the size differential between the query and the image subregion is large, and use Algorithm RED when the differential is small.

3.5 Another Metric for Histogram Distance

Other than the metric based on the Euclidean distance between the color histograms of the subimage query and the image subregion, another popular way to compute the similarity is based on the weighted Euclidean distance:

$$(Q - I)^T A (Q - I)$$

where A is a similarity matrix accounting for both the perceptual distance between the pairs of colors and the difference in the amounts of each color. Let d_{ij} be the Euclidean distance between colors i and j in the chosen color space (such as Luv and Munsell), then there are several choices for the entries a_{ij} in the matrix A [Sawhney and Hafner 1993]. For example, a_{ij} can be defined as (a) $1 - \frac{d_{ij}}{\max d_{ij}}$, (b) $e^{-\frac{\sigma d_{ij}}{\max d_{ij}}}$, and (c) $e^{-\sigma\left(\frac{d_{ij}}{\max d_{ij}}\right)^2}$, where σ is a positive constant.

With this metric, new Padding and Reduction Approaches can be stated as:

- New Padding Approach

$$\begin{aligned}
& \text{Given } \sum_{j=1}^{n} Q_j = v \text{ and } \sum_{j=1}^{n} I_j = w \geq v, \\
& \text{find an optimal vector } P^* \text{ to} \\
\textit{objective function} \quad & \text{minimize } (P+Q-I)^T A (P+Q-I) \qquad (3.9) \\
\textit{inequality constraint} \quad & \text{subject to } 0 \leq P_j \leq I_j \text{ for } 1 \leq j \leq n \\
\textit{summation constraint} \quad & \text{and } \sum_{j=1}^{n} P_j = w - v \\
\textit{domain constraint} \quad & \text{and } I, P, Q \text{ integer vectors}
\end{aligned}$$

- New Reduction Approach

$$\begin{aligned}
&\text{Given } \sum_{j=1}^{n} Q_j = v \text{ and } \sum_{j=1}^{n} I_j = w \geq v, \\
&\text{find an optimal vector } I'^{*} \text{ to} \\
\textit{objective function} \quad &\text{minimize } (Q - I')^T A (Q - I') \qquad (3.10) \\
\textit{inequality constraint} \quad &\text{subject to } 0 \leq I'_j \leq I_j \text{ for } 1 \leq j \leq n \\
\textit{summation constraint} \quad &\text{and } \sum_{j=1}^{n} I'_j = v \\
\textit{domain constraint} \quad &\text{and } I, I', Q \text{ integer vectors}
\end{aligned}$$

With an information preserving transformation such as Singular Value Decomposition, the similarity matrix A can be factorized into $B^T \Lambda B$ or $(\sqrt{\Lambda}B)^T(\sqrt{\Lambda}B)$ where Λ is a diagonal matrix. So, given $\sum_{j=1}^{n} Q_j = v$ and $\sum_{j=1}^{n} I_j = w \geq v$, the objective function of Problem (3.9) can be rewritten as:

$$\min_{P} \sum_{j=1}^{n} \lambda_{jj} \left[(BP)_j - (B\alpha)_j \right]^2 \qquad (3.11)$$

or

$$\min_{P} \sum_{j=1}^{n} \left[(\sqrt{\Lambda}BP)_j - (\sqrt{\Lambda}B\alpha)_j \right]^2 \qquad (3.12)$$

where $\alpha = I - Q$ and λ_{jj} is the j-th diagonal entry of the matrix Λ. Similarly, the objective function of Problem (3.10) can be rewritten as:

$$\min_{I'} \sum_{j=1}^{n} \lambda_{jj} \left[(BI')_j - (BQ)_j \right]^2 \qquad (3.13)$$

or

$$\min_{I'} \sum_{j=1}^{n} \left[(\sqrt{\Lambda}BI')_j - (\sqrt{\Lambda}BQ)_j \right]^2 \qquad (3.14)$$

With these representations, each of the two objective functions is in the form of the sum of squares of the difference terms. However, these difference terms are no longer monotonic, they are dependent on the values of entries in matrices B and Λ. So, increasing

the value of an entry in P or in I' may not reduce the value of the difference terms; sometimes, it may even boost the difference in these terms. Therefore, for this metric based on the weighted Euclidean distance, it is not easy to develop efficient tailor-made algorithms (such as Algorithms PAD and RED for the metric based on the Euclidean distance).

3.6 Summary

The histogram distance (D_H) between the histogram Q of the subimage query and the histogram I of the equal-sized image subregion can be computed by $D_H = (Q-I)^T(Q-I)$. However, given a subimage query of arbitrary size, the image subregion represented by the precomputed feature vector may not necessarily contain the same number of pixels as the query. For instance, it is not unusual that the precomputed vector of the image subregion contains more pixels than that of the query. As such, two algorithms — Algorithm PAD and Algorithm RED — have been developed for estimating the best-case lower bound ($\widehat{D_H}$) to the histogram distance. It has been shown that both algorithms give the same best-case lower bound. More precisely, in the domain of integers, given Q (where $\sum Q_j = v$) and I (where $\sum I_j = w \geq v$), the $\widehat{D_H}$ can be computed using Algorithm PAD:

$$\begin{aligned} \widehat{D_H} &= \min_P \sum_{j=1}^{n} (P_j - I_j + Q_j)^2 \\ &\text{s.t. } \forall j,\ 0 \leq P_j \leq I_j \text{ and } \sum_{j=1}^{n} P_j = w - v \end{aligned}$$

where P is the histogram of the padded area; the same $\widehat{D_H}$ can also be computed using Algorithm RED:

$$\begin{aligned} \widehat{D_H} &= \min_{I'} \sum_{j=1}^{n} (I'_j - Q_j)^2 \\ &\text{s.t. } \forall j,\ 0 \leq I'_j \leq I_j \text{ and } \sum_{j=1}^{n} I'_j = v \end{aligned}$$

where I' is the histogram of the reduced image subregion. In terms of performance, it is more efficient to use Algorithm PAD when the size differential between the subimage query and the image subregion is large, and to use Algorithm RED when the differential is small.

Chapter 4

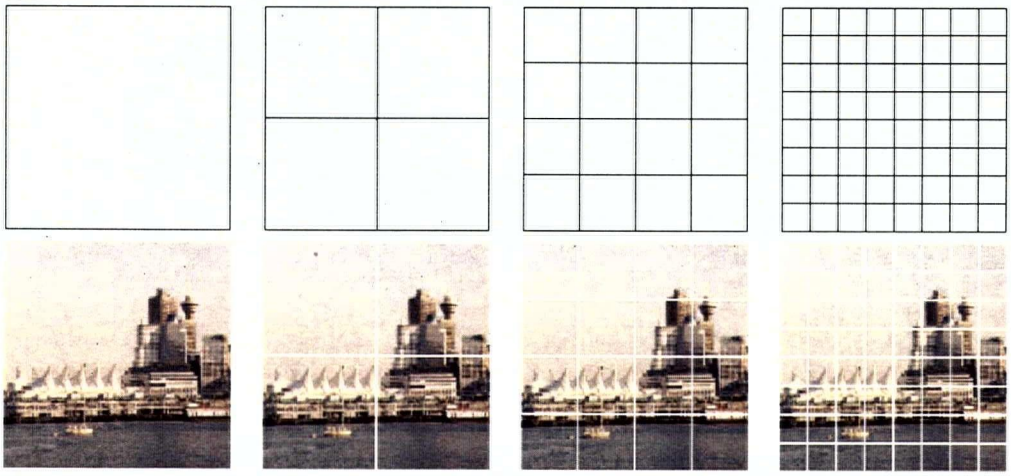
Multiscale Representation

In some IDBMSs, images are divided into blocks of a chosen size. For some queries, the scale at which the images are blocked may be too fine, and applying similarity comparisons to all those fine blocks may be a waste of effort. However, for some other queries, the scale may be too coarse, and the desired images may not be discriminated sufficiently. Given that subimage queries can be of arbitrary size, picking one best scale for all queries is hard, if not impossible. To cope with the challenge, we propose a representation that has multiple scales for matching. We also determine analytically and experimentally the appropriate number of levels for such a representation.

4.1 A 4-level Multiscale Representation

To process a user query, some IDBMSs divide each database image into blocks of a chosen size. For subimage queries of arbitrary size, picking a best scale at which the images are blocked is not easy. One way to cope with this challenge is to have multiple scales for matching. The idea is that depending on the scale or the need of a given query, a more appropriate scale can be used. With the multiscale representation, given any subimage query of arbitrary size, there exists an image subregion whose size is not smaller than the query. So, Padding and Reduction Algorithms can be applied in similarity matching. In this thesis, a 4-level multiscale representation (Figure 4.1) is proposed as follows:

- At Level H, the entire image is represented by a single global color histogram.
- At Level I, the image is divided into four non-overlapping blocks, and each block is represented by a color histogram covering $\frac{1}{4}$ of the entire image.
- At Level J, each block at Level I is further divided into four blocks, each of which is represented by a color histogram covering $\frac{1}{16}$ of the entire image.
- At Level K, each block at Level J is again divided into four, each of which is represented by a color histogram covering $\frac{1}{64}$ of the entire image.



(a) Level H (b) Level I (c) Level J (d) Level K

Figure 4.1: The 4-level Multiscale Representation

Given four levels of blocks for each image, many multi-level filtering schemes for finding u desired images (where u is the number of images requested by the user) can be developed. These include:

- a complete four-level HIJK scheme;
- three-level HIJ, HIK, HJK, and IJK schemes;
- two-level HI, HJ, HK, IJ, IK, and JK schemes; and
- one-level H, I, J, and K schemes.

Depending on the size of the subimage query and the number of levels intended to be examined, appropriate filtering schemes can be chosen. For example, given a subimage query whose size is smaller than $\frac{1}{16}$ of the entire image, the Scheme HJ can be applied in conjunction with Padding and Reduction Algorithms. With this two-level scheme, color histograms at Level I are skipped, and only the histograms at Levels H and J are considered.

The filtering scheme determines the levels (of color histograms representing the image subregions) which are intended to be considered, but it does not determine the order in which the histograms are to be examined. For example, it is unclear whether the histograms are to be checked on an image-by-image basis (with the use of vertical filter), on a level-by-level basis (with the use of horizontal filter), or in some other order. The search order is determined by the strategy for searching the multiscale representation. This will be examined in Chapter 5.

4.2 Formulation of Distance Function

A primary purpose of an IDBMS is to provide an environment for an efficient and effective retrieval of desired images. Given a large IDBMS, a key to the efficiency and the effectiveness of the search strategy for such retrievals relies on the formulation of the distance function. For instance, if the distance value at a coarser scale can be served as a lower bound to the distance value at a finer scale, then an efficient strategy with the use of vertical filters for searching the multiscale representation is possible.

4.2.1 Histogram Distance

The histogram distance (D_H) between the histogram Q of the subimage query and the histogram I of the equal-sized image subregion can be computed by $D_H = (Q-I)^T(Q-I)$. However, given that the arbitrary-size subimage query may not necessarily be of the same size as the image subregion, the best-case lower bound ($\widehat{D_H}$) to the histogram distance can be estimated using Algorithm PAD or Algorithm RED. More details can be found in Subsection 3.2.3.

Observation 4.1 Given that an image subregion $\mathtt{I}^{\mathtt{sup}}$ encloses another image subregion $\mathtt{I}^{\mathtt{sub}}$, and that the subimage query $\mathtt{Q}$ is smaller in size than $\mathtt{I}^{\mathtt{sub}}$ (Figure 4.2), the

estimated best-case histogram distance ($\widehat{D_H}$) between the histograms of $\mathtt{Q}$ and $\mathtt{I}^{\mathtt{sub}}$ is not smaller than the $\widehat{D_H}$ between the histograms of $\mathtt{Q}$ and $\mathtt{I}^{\mathtt{sup}}$:

$$\widehat{D_H}(\mathtt{Q}, \mathtt{I}^{\mathtt{sup}}) \leq \widehat{D_H}(\mathtt{Q}, \mathtt{I}^{\mathtt{sub}})$$

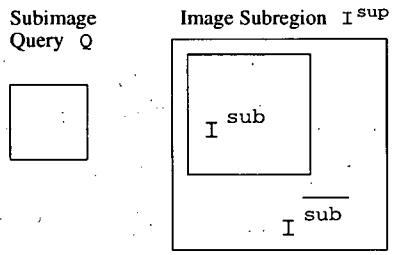


Figure 4.2: Subimage Query $\mathtt{Q}$ and Image Subregion $\mathtt{I}^{\mathtt{sup}}$

Explanation In addition to the pixels of $\mathtt{I}^{\mathtt{sub}}$, the image subregion $\mathtt{I}^{\mathtt{sup}}$ also contains the pixels found in $\mathtt{I}^{\overline{\mathtt{sub}}}$. So, for Algorithms PAD and RED, the selection of pixels in the minimization of $\widehat{D_H}(\mathtt{Q}, \mathtt{I}^{\mathtt{sup}})$ is at least the selection of pixels in the minimization of $\widehat{D_H}(\mathtt{Q}, \mathtt{I}^{\mathtt{sub}})$. The more the pixels selected, the smaller the $\widehat{D_H}$. ■

In other words, the $\widehat{D_H}$ between $\mathtt{Q}$ and $\mathtt{I}^{\mathtt{sup}}$ serves as a lower bound to the $\widehat{D_H}$ between $\mathtt{Q}$ and $\mathtt{I}^{\mathtt{sub}}$.

4.2.2 Positional Distance

The estimated best-case lower bound to the histogram distance measures the difference between the statistical distributions of various colors of the subimage query and the image subregion. However, a metric to measure the difference between the spatial locations of the subimage query and the image subregion is lacking. For instance, given a user query containing the central tower of the BC Legislative Buildings (Figure 4.3), the best-case lower bound to the histogram distance between the histograms of the Query and Image 1 is the same as that between the histograms of the Query and Image 2. However, in terms

of color and spatial location, Image 2 is clearly a better match. Therefore, an additional metric for measuring the positional difference between the subimage query and the image subregion is needed for better selectivity.

(a) Query (b) Image 1 (c) Image 2

Figure 4.3: The BC Legislative Buildings

There are a variety of ways to compute the positional distance between the query and the image subregion. Among them, some are more precise and complex than the others. It is well known that human memory is weak in retaining a fine granularity of spatial information of color; more often, the user may remember only an approximate location of the subimage when specifying his query. Moreover, when using Algorithm PAD or RED for handling subimage queries of arbitrary size, the best-case histogram distance is estimated, but the exact location of the padded query or the reduced image subregion is unknown. So, instead of computing the exact positional distance (D_P), we estimate its lower bound ($\widehat{D_P}$). The idea is based on the square of the shortest Euclidean distance between the original subimage query and the original non-overlapping image subregion. If the original query and the original image subregion overlap, the $\widehat{D_P}$ is defined to be 0.

Definition 4.1 Let q and i denote points in the subimage query and the image subregion respectively. Then, the estimated positional distance ($\widehat{D_P}$) is computed by:

$$\widehat{D_P} = \begin{cases} 0 & \text{when query and image subregion overlap} \\ \min ||q - i||^2 & \text{otherwise} \end{cases} \tag{4.1}$$

■

Let (q_x^{max}, q_y^{max}) and (q_x^{min}, q_y^{min}) denote the maximum and the minimum (x, y)-coordinates of the subimage query, and let (i_x^{max}, i_y^{max}) and (i_x^{min}, i_y^{min}) denote the maximum and the minimum (x, y)-coordinates of the image subregion. With these coordinates, the x-directional distance (dx) as well as the y-directional distance (dy) can be calculated, and the positional distance can then be estimated.

Algorithm 4.1 (Estimated Positional Distance)

1 $dx \leftarrow 0$
2 $dy \leftarrow 0$
3 **if** $q_x^{min} > i_x^{max}$
4 **then** $dx \leftarrow q_x^{min} - i_x^{max}$
5 **else if** $i_x^{min} > q_x^{max}$
6 **then** $dx \leftarrow i_x^{min} - q_x^{max}$
7 **if** $q_y^{min} > i_y^{max}$
8 **then** $dy \leftarrow q_y^{min} - i_y^{max}$
9 **else if** $i_y^{min} > q_y^{max}$
10 **then** $dy \leftarrow i_y^{min} - q_y^{max}$
11 **return** $(dx)^2 + (dy)^2$

Observation 4.2 Given that an image subregion $\mathtt{I}^{\mathtt{sup}}$ encloses another image subregion $\mathtt{I}^{\mathtt{sub}}$, and that the subimage query $\mathtt{Q}$ is smaller in size than $\mathtt{I}^{\mathtt{sub}}$, the estimated

positional distance ($\widehat{D_P}$) between $\mathtt{Q}$ and $\mathtt{I}^{\mathtt{sub}}$ is not shorter than the $\widehat{D_P}$ between $\mathtt{Q}$ and $\mathtt{I}^{\mathtt{sup}}$:

$$\widehat{D_P}(\mathtt{Q}, \mathtt{I}^{\mathtt{sup}}) \leq \widehat{D_P}(\mathtt{Q}, \mathtt{I}^{\mathtt{sub}})$$

Explanation The area of the image subregion $\mathtt{I}^{\mathtt{sup}}$ is the sum of the areas of image subregions $\mathtt{I}^{\mathtt{sub}}$ and $\mathtt{I}^{\overline{\mathtt{sub}}}$. So, the (x, y)-directional distances dx and dy between $\mathtt{Q}$ and $\mathtt{I}^{\mathtt{sup}}$ is not longer than those between $\mathtt{Q}$ ad $\mathtt{I}^{\mathtt{sub}}$. ■

In other words, the $\widehat{D_P}$ between $\mathtt{Q}$ and $\mathtt{I}^{\mathtt{sup}}$ serves as a lower bound to the $\widehat{D_P}$ between $\mathtt{Q}$ and $\mathtt{I}^{\mathtt{sub}}$.

4.2.3 Distance Function

To incorporate both the histogram distance D_H and the positional distance D_P, the distance between query $\mathtt{Q}$ and image subregion $\mathtt{I}$ is defined as a weighted sum of the form:

$$D = \beta D_H + (1 - \beta) D_P \tag{4.2}$$

where the parameter β is the user preference (with value ranging from 0 to 1) so that the user is allowed to specify the relative importance of the histogram distance and the positional distance. Given that the subimage query can be of arbitrary size, the D_H and the D_P can be estimated using the appropriate Algorithms described earlier, and the resulting estimated distance value ($\widehat{D}$) can be computed as the weighted sum of the estimated histogram distance and the estimated positional distance:

$$\widehat{D} = \beta \widehat{D_H} + (1 - \beta) \widehat{D_P} \tag{4.3}$$

Theorem 4.1 Given that an image subregion $\mathtt{I}^{\mathtt{sup}}$ encloses another image subregion $\mathtt{I}^{\mathtt{sub}}$, the estimated distance value ($\widehat{D}$) of the $\mathtt{Q}$-$\mathtt{I}^{\mathtt{sub}}$ pair (where the subimage query $\mathtt{Q}$ is smaller in size than $\mathtt{I}^{\mathtt{sub}}$) is greater than or equal to the $\widehat{D}$ of the $\mathtt{Q}$-$\mathtt{I}^{\mathtt{sup}}$ pair:

$$\widehat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{sup}}) \leq \widehat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{sub}})$$

Proof It is observable that $\widehat{D_H}(\mathtt{Q}, \mathtt{I}^{\mathtt{sup}}) \leq \widehat{D_H}(\mathtt{Q}, \mathtt{I}^{\mathtt{sub}})$ and that $\widehat{D_P}(\mathtt{Q}, \mathtt{I}^{\mathtt{sup}}) \leq \widehat{D_P}(\mathtt{Q}, \mathtt{I}^{\mathtt{sub}})$. So, for $0 \leq \beta \leq 1$, we can deduce:

$$\begin{aligned}
\hat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{sup}}) &= \beta\, \widehat{D_H}(\mathtt{Q}, \mathtt{I}^{\mathtt{sup}}) + (1-\beta)\, \widehat{D_P}(\mathtt{Q}, \mathtt{I}^{\mathtt{sup}}) \\
&\leq \beta\, \widehat{D_H}(\mathtt{Q}, \mathtt{I}^{\mathtt{sub}}) + (1-\beta)\, \hat{D}_P(\mathtt{Q}, \mathtt{I}^{\mathtt{sub}}) \\
&= \hat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{sub}})
\end{aligned}$$

In other words, the estimated distance value increases as the scale becomes finer. ■

Let $\mathtt{I}^{\mathtt{H}}, \mathtt{I}^{\mathtt{I}}, \mathtt{I}^{\mathtt{J}}$, and $\mathtt{I}^{\mathtt{K}}$ denote an image subregion at each of Levels H, I, J, and K such that $\mathtt{I}^{\mathtt{K}} \subseteq \mathtt{I}^{\mathtt{J}} \subseteq \mathtt{I}^{\mathtt{I}} \subseteq \mathtt{I}^{\mathtt{H}}$. According to Theorem 4.1, for a subimage query $\mathtt{Q}$ of size smaller than $\mathtt{I}^{\mathtt{I}}$, the relationship $\hat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{H}}) \leq \hat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{I}})$ holds. In terms of size, if such $\mathtt{Q}$ is smaller than $\mathtt{I}^{\mathtt{J}}$, then $\hat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{I}}) \leq \hat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{J}})$. Similarly, $\hat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{J}}) \leq \hat{D}(\mathtt{Q}, \mathtt{I}^{\mathtt{K}})$ provided that such $\mathtt{Q}$ is smaller than $\mathtt{I}^{\mathtt{K}}$. With the property that "the estimated distance keeps increasing (to its exact value) as the scale becomes finer", an efficient strategy with the use of vertical filters for searching multiscale representation is possible. The idea is that the distance value at the coarser scale serves as a lower bound to the distance value at the finer scale. So, poor matches having "large" distance values at the coarser scale (greater than the finest-scale distance values of the top-u [1]) can be eliminated without further computations at finer scales. For example, if the estimated Level-K distance for each of the top u images is below 200, a "poor match" with an estimated distance of 250 at Level I can be eliminated. The reason is that the estimated Level-K distance for this "poor match" is at least 250 (worse than those of the top-u). Without the property mentioned above, the search strategy needs to estimate the distance values at Levels J and K for this "poor match".

[1] As mentioned earlier in this chapter, u is the number of images requested by the user.

4.3 Multiscale Pure Vertical Search

Having formulated the distance function for the proposed 4-level representation, we can explore efficient and effective search strategies for retrieving the top u images from a database of M images. One of the search strategies is Search PV (Pure Vertical). In it, a vertical filter searches each image "vertically" across scales. The idea is that all M images are checked one after another. At any point in time, the current u smallest distances (at the finest scale of the u images, where $u \ll M$) are kept. When an image is tested, if its distance value at the current scale is already greater than some distance value of the current u smallest, the tested image can be eliminated. Otherwise, a finer scale is used. The process repeats until the image is (1) eliminated or (2) added to become one of the current u smallest (when it reaches the finest scale).

Definition 4.2 (Search PV) To conduct the Pure Vertical Search for retrieving the top u images from a collection of M images:

1. The first u images are checked at all levels/scales, and a distance value is computed for each image at the finest scale. These u images become the current top-u.
2. For each of the remaining $M - u$ images:
   (a) Let the coarsest scale be the current level/scale.
   (b) Compute the distance value. If the value is less than some value of the top-u, extend one level/scale (provided that we have not reached the finest scale) and repeat Step 2(b) on the extended scale. ■

Example 4.1 In many database applications, it is not unusual to retrieve the desired images from a collection of thousands of images. For simplicity, in this example, we try to find the top two images from a collection of seven images using Search PV (with Scheme HIJ). In the following trace, the number at the slot representing an image at a

particular level is the estimated distance value, and the superscript on its left denotes the search order.

To find the top two images using Search PV

	Img 1	Img 2	Img 3	Img 4	Img 5	Img 6	Img 7
Level H	^{1st} 25	^{4th} 5	^{7th} 65	^{9th} 80	^{10th} 10	^{13th} 70	^{14th} 68
Level I	^{2nd} 35	^{5th} 30	^{8th} 200		^{11th} 15		
Level J	^{3rd} 67	^{6th} 32			^{12th} 20		
	⇓	⇓	⇓	⇓	⇓	⇓	⇓
Top two images	Img 1	Imgs 2 and 1	Imgs 2 and 1	Imgs 2 and 1	Imgs 5 and 2	Imgs 5 and 2	Imgs 5 and 2

■

Since the histograms are examined image by image, the best way to organize the pre-computed feature vectors in the data file of the 4-level representation is to arrange them on an image-by-image basis. More precisely, the histograms associated with one image are followed by the histograms associated with another image. For each image, the 4 Level-I histograms are preceded by the Level-H histogram, succeeded by the 16 Level-J histograms, and the 64 Level-K histograms follow. Therefore, the file organization is of the form:

$$\underbrace{I^{\mathrm{H}}\, I^{\mathrm{I}}{}_{\mathrm{S}}\, I^{\mathrm{J}}{}_{\mathrm{S}}\, I^{\mathrm{K}}{}_{\mathrm{S}}}_{\text{for Image 1}} \quad \underbrace{I^{\mathrm{H}}\, I^{\mathrm{I}}{}_{\mathrm{S}}\, I^{\mathrm{J}}{}_{\mathrm{S}}\, I^{\mathrm{K}}{}_{\mathrm{S}}}_{\text{for Image 2}} \quad \cdots \quad \underbrace{I^{\mathrm{H}}\, I^{\mathrm{I}}{}_{\mathrm{S}}\, I^{\mathrm{J}}{}_{\mathrm{S}}\, I^{\mathrm{K}}{}_{\mathrm{S}}}_{\text{for Image } M}}$$

4.3.1 Analytical Evaluation

Cost Models

To measure the efficiency of Search PV, we set up cost models to estimate the CPU and the I/O costs. The CPU cost depends mainly on the time required to apply Padding and Reduction Algorithms to the data (color histograms), and the computation time for each

histogram (denoted by T_C) can be estimated using the experimental results in Section 3.4. The I/O cost depends mainly on the time required to sequentially or randomly access the pages containing the data (color histograms), and this access time can be affected by the size of data as well as the number of buffers. Given a minimum buffer (a buffer size of one page) and an intelligent buffer management scheme, the number of page accesses can be estimated with the use of statistical formulations.

Assuming that the value (pixel count) in each dimension of a color histogram requires 4 bytes, a total of $4n$ bytes are needed for one n-dimensional histogram. Hence, the total number of pages occupied by one histogram is $\frac{4n}{P}$ pages where P is the page size. In the cost models, color histograms at each level for each image are treated as a "record" [2], and the four "records" associated with each image can be grouped to form a "mega-record". So, for a database containing M images, there exists a total of $4M$ "records" or M "mega-records" for the 4-level representation. Depending on the level at which the histograms are represented, the size of "record" may vary. Sometimes a "record" may occupy a small portion of one page. For example, with a page size of 1 kilobyte (KB), the "record" of one Level-H 8-dimensional color histogram takes less than 4% of a page. With the file organization for Search PV, after the page containing this Level-H histogram is loaded, no extra page access is needed for reading its neighboring "record" (of the four Level-I histograms of the same image). However, sometimes a "record" may occupy more than one page. For example, with the above page size, the "record" of sixteen Level-J 512-dimensional color histograms takes more than 30 pages. The cost of accessing all the histograms in this "record" is the sum of the time required to randomly access the first page and the time required to sequentially access the remaining pages.

For a vast majority of subimage queries, in the case that the image subregions $\mathtt{I}^{\mathtt{sub}}$ and $\mathtt{I}^{\mathtt{sup}}$ are the best matches (which give the smallest $\hat{D}s$) at their corresponding levels,

[2] For example, the four Level-I histograms for Image 1 form a "record", and the sixteen Level-J histograms for Image 3 form another "record".

$\mathtt{I}^{\mathtt{sub}}$ is one of the subregions enclosed in $\mathtt{I}^{\mathtt{sup}}$. When aiming for a balance of efficiency and effectiveness, checking all the histograms in a "record" may seem unnecessary, and only a portion of the "record" (some of the histograms) may need to be examined. For instance, with Scheme IJ, the most promising image subregion $\mathtt{I}^{\mathtt{I}}$ can be found after checking histograms of the four subregions at Level I. Then, rather than considering all sixteen Level-J histograms, we consider the four Level-J histograms which represent the subregions covered by $\mathtt{I}^{\mathtt{I}}$. As a result, both the CPU and the I/O costs can be reduced.

Example 4.2 For each image in Scheme HIJ, we examine the Level-H histogram, and then consider (if necessary [3]) the 4 Level-I histograms and (if necessary [4]) the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I. Hence, using a 64-dimensional color histogram with page size $P = 1$ KB, the cost model for Scheme HIJ can be described as follows:

- The CPU cost is the product of T_C and the total number of histograms used in the computation. In Scheme HIJ, Level-H histograms are examined for all M images, 4 Level-I histograms are checked for each of the ξ_4 images, and 4 Level-J histograms are examined for each of the ξ_5 images (where $M \geq \xi_4 \geq \xi_5 \geq u$, and both ξ_4 and ξ_5 are determined dynamically at runtime). Hence, $CPU^{PV}_{HIJ} = (M + 4\xi_4 + 4\xi_5)T_C$.

- The I/O cost is the sum of total seek times and total data transfer times. Using the above file organization for Search PV, an average seek time (denoted by T_A) is charged for moving the read head to the first "record". Since all M images are searched one after another and their "records" are stored contiguously on an image-by-image basis, only a minimum seek time (denoted by T_M) is charged for each jump between the data (for example, jumping from the fourth Level-I histogram to the

[3] We consider the 4 Level-I histograms of an image if the $\widehat{D}$ at Level H is less than any $\widehat{D}$ of the current top-u.

[4] We consider the 4 Level-J histograms of an image if the best $\widehat{D}$ at Level I is less than any $\widehat{D}$ of the current top-u.

third Level-J histogram of the same image, and jumping from the Level-J "record" of an image to the Level-H "record" of the next image). Moreover, when a page of data (color histograms) is loaded, a data transfer time (denoted by T_D) is charged.

More precisely, for Scheme HIJ, T_A is needed to get to the Level-H histogram of the first image, and T_D is charged for loading the page containing the histogram. Since only 4 histograms can fit into a page, if the 4 Level-I histograms happen to be examined, then we load the next page. There is a probability of $\frac{1}{4}$ that the selected 4 Level-J histograms are stored contiguously after the loaded Level-I histograms. Hence, $I/O_{HIJ}^{PV} = T_A + (M - 1 + \frac{3\xi_5}{4})T_M + (M + \xi_4 + \xi_5)T_D$.

The cost models for other filtering schemes, color histograms of other dimensions, or other page sizes can be formulated in a similar manner. ■

The cost models for the 15 filtering schemes (with page size $P = 1$ KB) are shown below:

1. Scheme H

   For each image, we examine the Level-H histogram.

   - $CPU_H^{PV} = MT_C$
   - $I/O_H^{PV} = T_A + (M-1)T_M + MT_D$ for 8-dimensional (8-D) histograms

     $I/O_H^{PV} = T_A + (M-1)T_M + MT_D$ for 64-dimensional (64-D) histograms

     $I/O_H^{PV} = T_A + (M-1)T_M + 2MT_D$ for 512-dimensional (512-D) histograms

2. Scheme I

   For each image, we examine the 4 Level-I histograms.

   - $CPU_I^{PV} = 4MT_C$
   - $I/O_I^{PV} = T_A + (M-1)T_M + MT_D$ for 8-D histograms

     $I/O_I^{PV} = T_A + (M-1)T_M + MT_D$ for 64-D histograms

     $I/O_I^{PV} = T_A + (M-1)T_M + 8MT_D$ for 512-D histograms

3. Scheme HI

   For each image, we examine the Level-H histogram, and if necessary [5], consider the 4 Level-I histograms.

   - $CPU_{HI}^{PV} = (M + 4\xi_1)T_C$
   - $I/O_{HI}^{PV} = T_A + (M-1)T_M + MT_D$ for 8-D histograms

     $I/O_{HI}^{PV} = T_A + (M-1)T_M + (M+\xi_1)T_D$ for 64-D histograms

     $I/O_{HI}^{PV} = T_A + (M-1)T_M + (2M+8\xi_1)T_D$ for 512-D histograms

   where $M \geq \xi_1 \geq u$.

4. Scheme J

   For each image, we examine the 16 Level-J histograms.

   - $CPU_J^{PV} = 16MT_C$
   - $I/O_J^{PV} = T_A + (M-1)T_M + MT_D$ for 8-D histograms

     $I/O_J^{PV} = T_A + (M-1)T_M + 4MT_D$ for 64-D histograms

     $I/O_J^{PV} = T_A + (M-1)T_M + 32MT_D$ for 512-D histograms

5. Scheme HJ

   For each image, we examine the Level-H histogram, and if necessary, consider the 16 Level-J histograms.

   - $CPU_{HJ}^{PV} = (M + 16\xi_2)T_C$
   - $I/O_{HJ}^{PV} = T_A + (M-1)T_M + MT_D$ for 8-D histograms

     $I/O_{HJ}^{PV} = T_A + (M-1+\xi_2)T_M + (M+4\xi_2)T_D$ for 64-D histograms

     $I/O_{HJ}^{PV} = T_A + (M-1+\xi_2)T_M + (2M+32\xi_2)T_D$ for 512-D histograms

   where $M \geq \xi_2 \geq u$.

[5] We consider the 4 Level-I histograms of an image if the $\widehat{D}$ at Level H is less than any $\widehat{D}$ of the current top-u. Similar conditions apply to the filtering schemes involving more than one level.

6. Scheme IJ

   For each image, we examine the 4 Level-I histograms, and if necessary, consider the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I.

   - $CPU_{IJ}^{PV} = (4M + 4\xi_3)T_C$
   - $I/O_{IJ}^{PV} = T_A + (M-1)T_M + MT_D$ for 8-D histograms

     $I/O_{IJ}^{PV} = T_A + (M - 1 + \frac{3\xi_3}{4})T_M + (M + \xi_3)T_D$ for 64-D histograms

     $I/O_{IJ}^{PV} = T_A + (M - 1 + \frac{3\xi_3}{4})T_M + (8M + 8\xi_3)T_D$ for 512-D histograms

   where $M \geq \xi_3 \geq u$.

7. Scheme HIJ

   For each image, we examine the Level-H histogram, then consider (if necessary) the 4 Level-I histograms and (if necessary) the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I.

   - $CPU_{HIJ}^{PV} = (M + 4\xi_4 + 4\xi_5)T_C$
   - $I/O_{HIJ}^{PV} = T_A + (M-1)T_M + MT_D$ for 8-D histograms

     $I/O_{HIJ}^{PV} = T_A + (M - 1 + \frac{3\xi_5}{4})T_M + (M + \xi_4 + \xi_5)T_D$ for 64-D histograms

     $I/O_{HIJ}^{PV} = T_A + (M - 1 + \frac{3\xi_5}{4})T_M + (2M + 8\xi_4 + 8\xi_5)T_D$ for 512-D histograms

   where $M \geq \xi_4 \geq \xi_5 \geq u$.

8. Scheme K

   For each image, we examine the 64 Level-K histograms.

   - $CPU_K^{PV} = 64MT_C$
   - $I/O_K^{PV} = T_A + (M-1)T_M + 2MT_D$ for 8-D histograms

     $I/O_K^{PV} = T_A + (M-1)T_M + 16MT_D$ for 64-D histograms

     $I/O_K^{PV} = T_A + (M-1)T_M + 128MT_D$ for 512-D histograms

9. Scheme HK

   For each image, we examine the Level-H histogram, and if necessary, consider the 64 Level-K histograms.

   - $CPU_{HK}^{PV} = (M + 64\xi_6)T_C$
   - $I/O_{HK}^{PV} = T_A + (M-1)T_M + (M + 2\xi_6)T_D$ for 8-D histograms

     $I/O_{HK}^{PV} = T_A + (M - 1 + \xi_6)T_M + (M + 16\xi_6)T_D$ for 64-D histograms

     $I/O_{HK}^{PV} = T_A + (M - 1 + \xi_6)T_M + (2M + 128\xi_6)T_D$ for 512-D histograms

   where $M \geq \xi_6 \geq u$.

10. Scheme IK

    For each image, we examine the 4 Level-I histograms, and if necessary, consider the 16 Level-K histograms which represent the subregions enclosed in the promising block of Level I.

    - $CPU_{IK}^{PV} = (4M + 16\xi_7)T_C$
    - $I/O_{IK}^{PV} = T_A + (M - 1 + \frac{3\xi_7}{4})T_M + (M + \xi_7)T_D$ for 8-D histograms

      $I/O_{IK}^{PV} = T_A + (M - 1 + \xi_7)T_M + (M + 4\xi_7)T_D$ for 64-D histograms

      $I/O_{IK}^{PV} = T_A + (M - 1 + \xi_7)T_M + (8M + 32\xi_7)T_D$ for 512-D histograms

    where $M \geq \xi_7 \geq u$.

11. Scheme JK

    For each image, we examine the 16 Level-J histograms, and if necessary, consider the 4 Level-K histograms which represent the subregions enclosed in the promising block of Level J.

    - $CPU_{JK}^{PV} = (16M + 4\xi_8)T_C$

- $I/O_{JK}^{PV} = T_A + (M - 1 + \frac{11\xi_8}{16})T_M + (M + \frac{3\xi_8}{4})T_D$ for 8-D histograms

  $I/O_{JK}^{PV} = T_A + (M - 1 + \frac{15\xi_8}{16})T_M + (4M + \xi_8)T_D$ for 64-D histograms

  $I/O_{JK}^{PV} = T_A + (M - 1 + \frac{15\xi_8}{16})T_M + (32M + 8\xi_8)T_D$ for 512-D histograms

where $M \geq \xi_8 \geq u$.

12. Scheme HIK

For each image, we examine the Level-H histogram, then consider (if necessary) the 4 Level-I histograms and (if necessary) the 16 Level-K histograms which represent the subregions enclosed in the promising block of Level I.

- $CPU_{HIK}^{PV} = (M + 4\xi_9 + 16\xi_{10})T_C$
- $I/O_{HIK}^{PV} = T_A + (M - 1 + \frac{3\xi_{10}}{4})T_M + (M + \xi_{10})T_D$ for 8-D histograms

  $I/O_{HIK}^{PV} = T_A + (M - 1 + \xi_{10})T_M + (M + \xi_9 + 4\xi_{10})T_D$ for 64-D histograms

  $I/O_{HIK}^{PV} = T_A + (M - 1 + \xi_{10})T_M + (2M + 8\xi_9 + 32\xi_{10})T_D$ for 512-D histograms

where $M \geq \xi_{10} \geq \xi_9 \geq u$.

13. Scheme HJK

For each image, we examine the Level-H histogram, then consider (if necessary) the 16 Level-J histograms and (if necessary) the 4 Level-K histograms which represent the subregions enclosed in the promising block of Level J.

- $CPU_{HJK}^{PV} = (M + 16\xi_{11} + 4\xi_{12})T_C$
- $I/O_{HJK}^{PV} = T_A + (M - 1 + \frac{13\xi_{12}}{16})T_M + (M + \frac{7\xi_{12}}{8})T_D$ for 8-D histograms

  $I/O_{HJK}^{PV} = T_A + (M - 1 + \xi_{11} + \frac{3\xi_{12}}{4})T_M + (M + 4\xi_{11} + \xi_{12})T_D$ for 64-D histograms

  $I/O_{HJK}^{PV} = T_A + (M - 1 + \xi_{11} + \frac{3\xi_{12}}{4})T_M + (2M + 32\xi_{11} + 8\xi_{12})T_D$

  for 512-D histograms

where $M \geq \xi_{11} \geq \xi_{12} \geq u$.

14. Scheme IJK

For each image, we examine the 4 Level-I histograms, then consider (if necessary) the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I, and consider (if necessary) the 4 Level-K histograms which represent the subregions enclosed in the promising block of Level J.

- $CPU_{IJK}^{PV} = (4M + 4\xi_{13} + 4\xi_{14})T_C$
- $I/O_{IJK}^{PV} = T_A + (M - 1 + \frac{3\xi_{14}}{4})T_M + (M + \frac{13\xi_{14}}{16})T_D$ for 8-D histograms

  $I/O_{IJK}^{PV} = T_A + (M - 1 + \frac{3\xi_{13}}{4} + \xi_{14})T_M + (M + \xi_{13} + \xi_{14})T_D$ for 64-D histograms

  $I/O_{IJK}^{PV} = T_A + (M - 1 + \frac{3\xi_{13}}{4} + \xi_{14})T_M + (8M + 8\xi_{13} + 8\xi_{14})T_D$

  for 512-D histograms

where $M \geq \xi_{13} \geq \xi_{14} \geq u$.

15. Scheme HIJK

For each image, we examine the Level-H histogram, then consider (if necessary) the 4 Level-I histograms, (if necessary) the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I, and (if necessary) the 4 Level-K histograms which represent the subregions enclosed in the promising block of Level J.

- $CPU_{HIJK}^{PV} = (M + 4\xi_{15} + 4\xi_{16} + 4\xi_{17})T_C$
- $I/O_{HIJK}^{PV} = T_A + (M - 1 + \frac{13\xi_{17}}{16})T_M + (M + \frac{7\xi_{17}}{8})T_D$ for 8-D histograms

  $I/O_{HIJK}^{PV} = T_A + (M - 1 + \frac{3\xi_{16}}{4} + \xi_{17})T_M + (M + \xi_{15} + \xi_{16} + \xi_{17})T_D$

  for 64-D histograms

  $I/O_{HIJK}^{PV} = T_A + (M - 1 + \frac{3\xi_{16}}{4} + \xi_{17})T_M + (2M + 8\xi_{15} + 8\xi_{16} + 8\xi_{17})T_D$

  for 512-D histograms

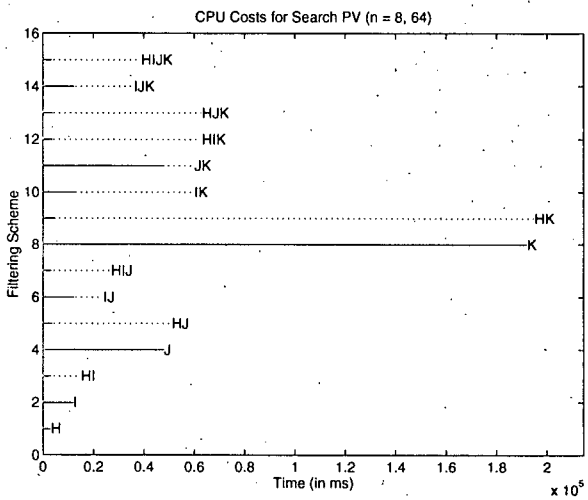
where $M \geq \xi_{15} \geq \xi_{16} \geq \xi_{17} \geq u$.

Notice from the cost models of the filtering schemes which involve more than one level, the value of each ξ_j is determined dynamically. For instance, in the best case of Scheme HIJ, the first u images are the top-u, and each Level-H distance for the remaining $M - u$ images is greater than all Level-J distances of the top u images. As such, the histograms at Levels I and J for the $M - u$ images are not examined; thus, $\xi_4 = \xi_5 = u$. Conversely, in the worst case, at any point in time the Level-H and Level-I distances of each image are less than some Level-J distance of the current top-u. As such, all the histograms at all levels for the M images are examined; thus, $\xi_4 = \xi_5 = M$.

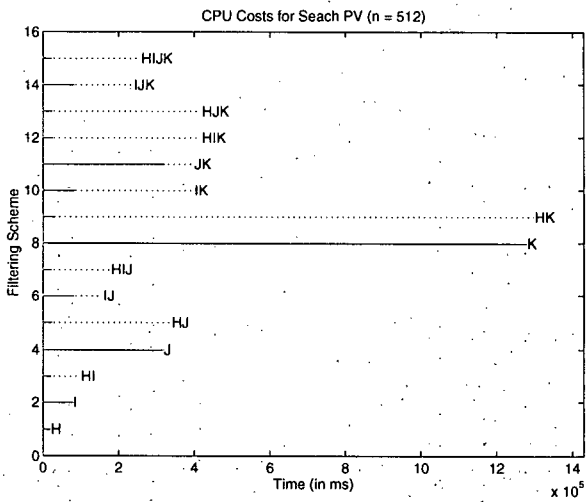
Analytical Results

The above cost models provide a good foundation for analyzing the 15 filtering schemes, and for discovering general trends. The computation time for Padding and Reduction Algorithms (T_C) depends on the sizes of the subimage query and the image subregion. For the analysis, we let T_C be 3 ms for the 8-dimensional and the 64-dimensional histograms, and 20 ms for the 512-dimensional histograms. We assume that each minimum seek (T_M) takes 5 ms, each average seek (T_A) requires 15 ms, and the transfer of data in each page (T_D) with page size P = 1 KB needs 0.3 ms. The analytical results for finding $u = 10$ images from a database of M=1000 images are summarized in the figures on the next few pages. In the figures, for each filtering scheme, the minimum cost is shown by the solid line, and the additional cost (if any) is indicated by the dotted line

- CPU Costs



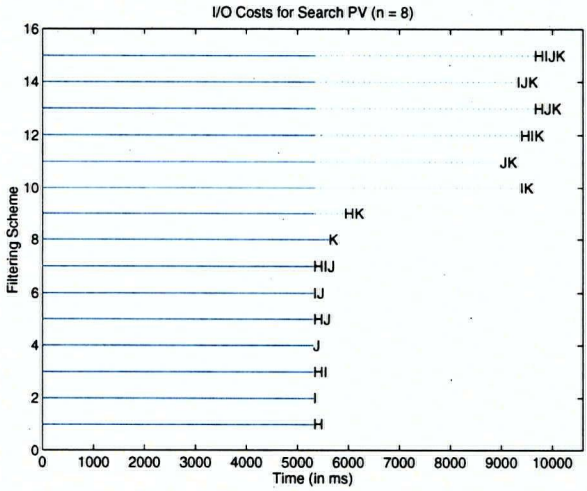
(a) 8-, 64-dimensional Color Histograms



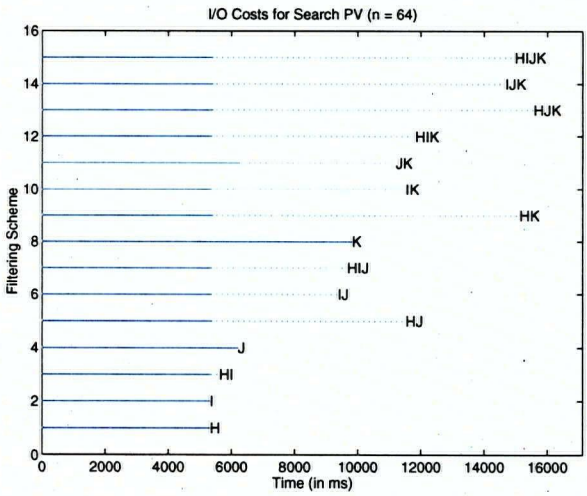
(b) 512-dimensional Color Histograms

Figure 4.4: CPU Costs for Search PV

- I/O Costs

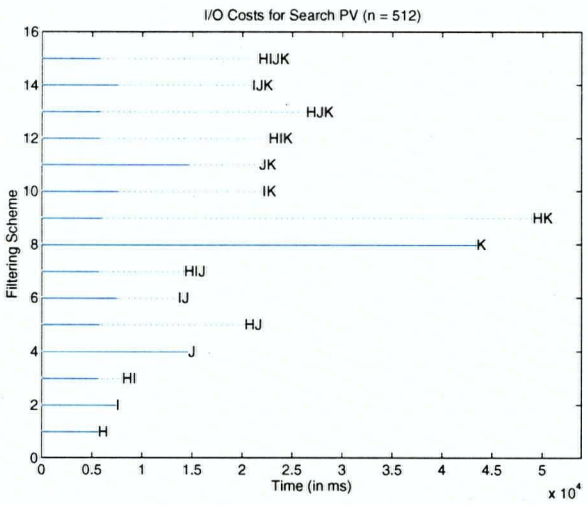


(a) 8-dimensional Color Histogram



(b) 64-dimensional Color Histogram

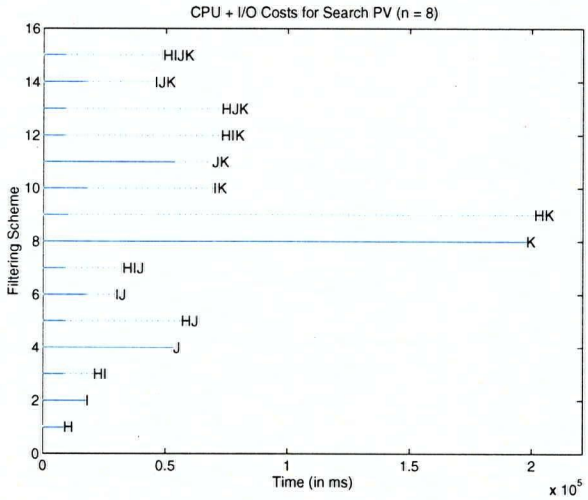
Figure 4.5: I/O Costs for Search PV



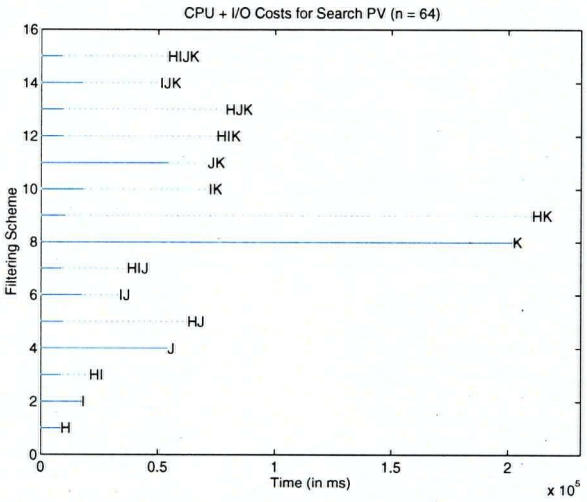
(c) 512-dimensional Color Histogram

Figure 4.5: I/O Costs for Search PV (Continued)

- Combined CPU and I/O Costs

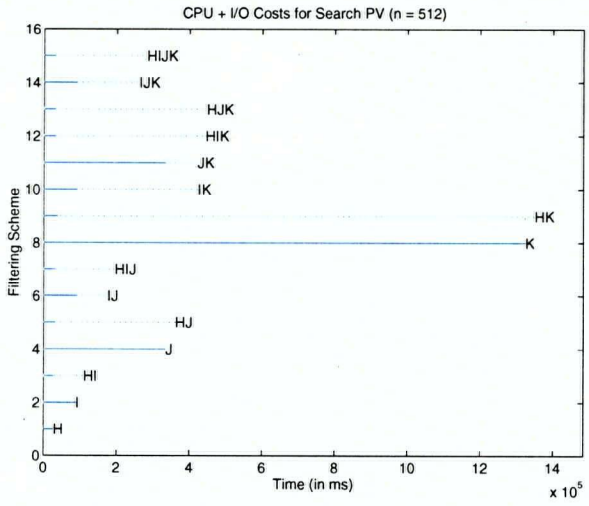


(a) 8-dimensional Color Histogram



(b) 64-dimensional Color Histogram

Figure 4.6: Combined CPU and I/O Costs for Search PV



(c) 512-dimensional Color Histogram

Figure 4.6: Combined CPU and I/O Costs for Search PV (Continued)

Observation 4.3 The CPU cost is affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required for CPU operations increases.

Explanation The CPU cost is proportional to the computation time T_C for Algorithms PAD and RED. Recall from Observation 3.1 that T_C increases as the dimension of the histogram grows. ■

Observation 4.4 For low dimensional (for example, 8-dimensional) color histograms, the I/O costs for many filtering schemes appear to be the same. For example, $I/O_H^{PV} = I/O_{HI}^{PV} = I/O_{HIJ}^{PV}$.

Explanation Each of the 8-dimensional histograms occupies less than 4% of a page. So, the Level-H, Level-I, and Level-J "records" associated with an image can fit into one page. As a result, after the Level-H "record" of histogram is loaded, no extra page access is needed for histograms contained in the Level-I and Level-J "records". ■

Observation 4.5 The I/O cost is also affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required for I/O operations increases.

Explanation The I/O cost depends on the number of pages occupied by the n-dimensional color histograms. As the dimension of the histograms grows, the number of pages occupied by one histogram ($= \frac{4n}{P}$ pages where P is the page size) increases, and thus the total number of pages occupied increases as well. Hence, the I/O cost increases.

For the same reason, as the page size P shrinks, the number of pages occupied by one histogram ($= \frac{4n}{P}$ pages) increases, and thus the time required for I/O operations also increases. ■

Observation 4.6 The combined CPU and I/O cost is affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required for both CPU and I/O operations increases.

Explanation Note from Observation 4.3 that the CPU cost increases as the dimension of the histogram grows, and from Observation 4.5 that the I/O cost also increases as the dimension of the histogram grows. Thus, the combined CPU and I/O cost follows the same trend. ■

Observation 4.7 Filtering schemes which start with filter J or K often take more CPU and I/O time. For example, the combined CPU and I/O times demanded by Schemes J, JK, and K are often longer than the times taken by the worst cases of many filtering schemes such as Schemes HIJ and HIJK.

Explanation For the scheme that starts with filter J, at least $16M$ histograms are required to be examined; for the scheme that starts with filter K, at least $64M$ histograms are required to be examined. These numbers (of histograms to be examined) are large when compared with at most $9M$ $(= M + 4M + 4M)$ histograms for Scheme HIJ and at most $13M$ $(= M + 4M + 4M + 4M)$ histograms for Scheme HIJK. ■

Observation 4.8 Filtering schemes which skip intermediate levels often incur greater CPU and I/O cost.

Explanation For the filtering scheme that does not skip any level, at the next (finer) scale, we consider the 4 histograms which represent the subregions enclosed in the most promising block of the current scale. If a filtering scheme skips one level, then we bypass the 4 histograms at the level skipped; however, at the next available scale, we need to consider 16 histograms, each of which covers a subregion enclosed in the promising block of the current scale. Similarly, if a filtering scheme skips two consecutive levels, then we bypass the 8 (= 4 + 4) histograms at the levels skipped. Unfortunately, at the next available scale, we need to consider 64 histograms, each of which covers a subregion enclosed in the promising block of the current scale. ■

Observation 4.9 The combined CPU and I/O cost for the additional intermediate levels becomes relatively less expensive as the dimension of the histogram grows. For example, as the dimension of the histogram increases, the combined cost for Scheme HJ is usually more expensive than that for Scheme HIJ. Similarly, the combined cost for Scheme HK is usually more expensive than that for Scheme HIK or HJK, and either of these two costs is more expensive than that for Scheme HIJK.

Explanation Following from the previous observation, if a filtering scheme skips one level, we need to consider an extra 12 (= 16 − 4) histograms, and if the filtering scheme skips two consecutive levels, we need to consider an extra 56 (= 64 − 8) histograms. As the dimension of the color histogram grows, fewer histograms fit into a page. Thus, the total number of pages occupied by the histograms increases. It is relatively more expensive to bypass the intermediate levels. ■

In general, the above analytical results tend to favor the filtering schemes which start with Level H (or Level I), and those that do not skip any intermediate level. More precisely, Schemes H, I, HI, IJ, HIJ, IJK, and HIJK are favored.

4.3.2 Experimental Evaluation

To find the appropriate filtering schemes for Search PV, several experiments have been performed using color histograms of 8, 64, and 512 dimensions. These color histograms are created for a database consisting of 1000 real images which are collected from various sources and stored in Vista formats [Pope and Lowe 1994]. Examples of these images are pictures of tourist attractions in British Columbia, photos of a water project in California, and scenic shots taken at different cities throughout the world.

As observed in Section 3.4, the computation time for Padding and Reduction Algorithms is affected by the sizes of the subimage query and the image subregion. In the experiments, we used 48 subimage queries of different sizes. These queries were classified into four main types, and we studied the performance/speed and the accuracy of each query type.

Definition 4.3 Given that subimage queries can be of arbitrary size, we classified them into 4 main types:

1. "h"-typed queries — whose sizes are larger than image subregions at Level I,
2. "i"-typed queries — whose sizes are smaller than image subregions at Level I but larger than image subregions at Level J,
3. "j"-typed queries — whose sizes are smaller than image subregions at Level J but larger than image subregions at Level K, and
4. "k"-typed queries — whose sizes are smaller than image subregions at Level K. ■

The execution time (denoted by T_E, and measuring the combined CPU&I/O time per query on a database of 1000 images) can be used in assessing the efficiency of Search PV. Several measures can be applied in assessing the effectiveness of the strategy. Examples are (1) fallout and (2) standard recall and precision [Salton and McGill 1983]. Fallout shows

the portion of non-relevant images being retrieved. Standard recall indicates the portion of the relevant images being retrieved, and standard precision indicates the portion of the retrieved images that are relevant. One problem with this set of measures is that images are categorized into relevant and non-relevant; however, the degree of relevancy cannot be expressed. Another example of a measure for assessing the effectiveness of the search strategy is a variant of normalized recall [Faloutsos *et al* 1994]. It calculates the ratio of the average rank of all retrieved relevant images to the ideal average rank. However, images may sometimes be so similar that they are almost indistinguishable; in such cases, it may not be easy to assign exact ranks to the images. Hence, for each subimage query, we categorize the database images into five main classes such that the best 2 images fall into Class A, the next 3 into Class B, and the next 5 into Class C. Then, 15 images ranked from 11-th to 25-th are categorized into Class D, while the remaining $M - 25$ images are categorized into Class E. All images in the same class have the same rank.

Definition 4.4 With the categorization of images into five classes, the effectiveness of the search strategy can be assessed by counting the number of retrieved images which fall into each class. A dissimilarity score (DS) for the retrieval of the best 10 images can be computed as a sum of the weighted difference:

$$DS = 7(2 - \eta_A) + 5(3 - \eta_B) + (5 - \eta_C) + 15\eta_D + 40\eta_E \tag{4.4}$$

where η_j indicates the number of images falling into Class j. In the ideal situation, the DS is 0. ■

The experiments were run on a Sun UltraSPARC-1 workstation using a page size of 1 KB and user preference β of 0.5 [6]. The results (the average DS and T_E values for the retrievals of the best 10 images from a collection of 1000 images) are summarized in

[6] With the user preference β set to 0.5, both the color distribution and spatial information are of the same level of importance.

the tables and figures on the next few pages. The T_E measures the efficiency and the DS assesses the effectiveness.

- "h"-typed queries

Filtering Scheme	Dimension of Color Histograms		
	8-dimensional	64-dimensional	512-dimensional
	η_A η_B η_C η_D	η_A η_B η_C η_D	η_A η_B η_C η_D
H	0.7 0.6 1.5 7.2	1.9 2.1 1.7 4.3	1.9 2.4 1.7 4.0
	$\Rightarrow DS = 132.6$	$\Rightarrow DS = 73.0$	$\Rightarrow DS = 67.0$
	$T_E = 2.4 \pm 1.1$ s	$T_E = 4.5 \pm 3.8$ s	$T_E = 43.4 \pm 22.5$ s

Table 4.1: Experimental Results for Search PV ("h"-typed Queries)

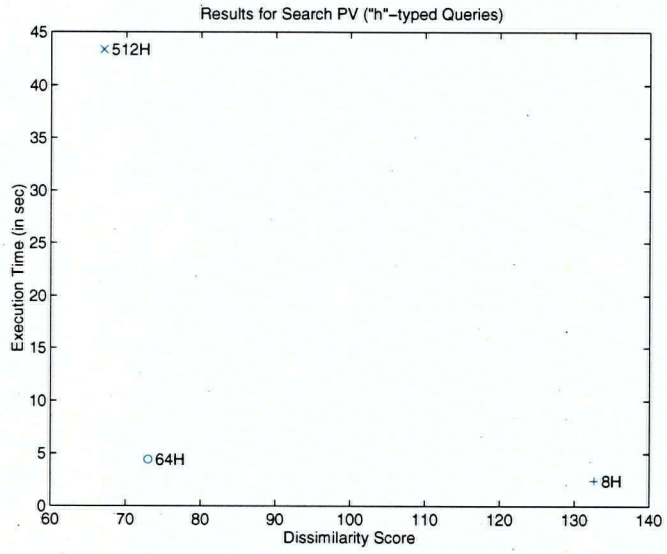


Figure 4.7: Experimental Results for Search PV ("h"-typed Queries)

Among the three dimensions of color histograms, the 8-dimensional case appears to require the shortest execution time, but it has the largest dissimilarity value. By contrast, the 512-dimensional case has the smallest dissimilarity value, but it requires the longest execution time. So, in keeping a balance of efficiency and effectiveness, the best filtering scheme for "h"-typed queries, when operated with Padding and Reduction Algorithms, is Scheme H with 64-dimensional color histograms.

- "i"-typed queries

Filtering Scheme	Dimension of Color Histograms		
	8-dimensional	64-dimensional	512-dimensional
	η_A η_B η_C η_D	η_A η_B η_C η_D	η_A η_B η_C η_D
H	0.3 0.9 1.3 7.5	1.5 2.2 1.8 4.5	1.9 2.4 1.4 4.3
	$\Rightarrow DS = 138.6$	$\Rightarrow DS = 78.2$	$\Rightarrow DS = 71.8$
	$T_E = 2.5 \pm 0.6$ s	$T_E = 5.5 \pm 1.5$ s	$T_E = 42.1 \pm 27.2$ s
I	0.5 0.8 1.3 7.4	1.8 2.1 1.8 4.3	1.9 2.4 1.5 4.2
	$\Rightarrow DS = 136.2$	$\Rightarrow DS = 73.6$	$\Rightarrow DS = 70.2$
	$T_E = 4.1 \pm 2.1$ s	$T_E = 9.2 \pm 8.5$ s	$T_E = 127.4 \pm 87.9$ s
HI	0.5 0.8 1.3 7.4	1.8 2.1 1.8 4.3	1.9 2.4 1.5 4.2
	$\Rightarrow DS = 136.2$	$\Rightarrow DS = 73.6$	$\Rightarrow DS = 70.2$
	$T_E = 2.7 \pm 0.6$ s	$T_E = 7.1 \pm 0.8$ s	$T_E = 60.4 \pm 27.8$ s

Table 4.2: Experimental Results for Search PV ("i"-typed Queries)

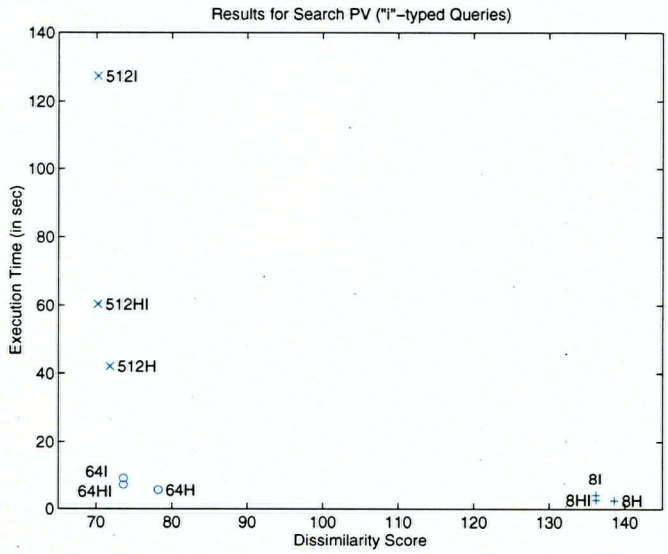


Figure 4.8: Experimental Results for Search PV ("i"-typed Queries)

To aim for a balance of performance and accuracy, the best filtering scheme for "i"-typed queries, as observed from the above experimental results, is Scheme HI with 64-dimensional color histograms.

- "j"-typed queries

Filtering Scheme	Dimension of Color Histograms		
	8-dimensional	64-dimensional	512-dimensional
	η_A η_B η_C η_D	η_A η_B η_C η_D	η_A η_B η_C η_D
H	0.0 0.8 1.2 8.0	0.9 2.1 2.5 4.5	1.5 2.5 1.2 4.8
	$\Rightarrow DS = 148.8$	$\Rightarrow DS = 82.2$	$\Rightarrow DS = 81.8$
	$T_E = 1.5 \pm 0.1$ s	$T_E = 2.6 \pm 0.2$ s	$T_E = 23.7 \pm 9.3$ s
I	0.0 1.0 1.3 7.7	1.5 2.2 1.8 4.5	1.7 2.3 1.5 4.5
	$\Rightarrow DS = 143.2$	$\Rightarrow DS = 78.2$	$\Rightarrow DS = 76.6$
	$T_E = 3.0 \pm 0.6$ s	$T_E = 8.5 \pm 1.6$ s	$T_E = 168.3 \pm 25.6$ s
HI	0.0 1.0 1.3 7.7	1.5 2.2 1.8 4.5	1.7 2.3 1.5 4.5
	$\Rightarrow DS = 143.2$	$\Rightarrow DS = 78.2$	$\Rightarrow DS = 76.6$
	$T_E = 1.5 \pm 0.1$ s	$T_E = 3.4 \pm 0.7$ s	$T_E = 36.5 \pm 16.7$ s
IJ	0.5 0.8 1.3 7.4	1.8 2.0 1.9 4.3	1.7 2.3 1.7 4.3
	$\Rightarrow DS = 136.2$	$\Rightarrow DS = 74.0$	$\Rightarrow DS = 73.4$
	$T_E = 3.4 \pm 0.5$ s	$T_E = 9.2 \pm 1.6$ s	$T_E = 188.0 \pm 37.4$ s
HIJ	0.5 0.8 1.3 7.4	1.8 2.0 1.9 4.3	1.7 2.3 1.7 4.3
	$\Rightarrow DS = 136.2$	$\Rightarrow DS = 74.0$	$\Rightarrow DS = 73.4$
	$T_E = 2.8 \pm 0.3$ s	$T_E = 5.8 \pm 1.7$ s	$T_E = 68.5 \pm 31.7$ s

Table 4.3: Experimental Results for Search PV ("j"-typed Queries)

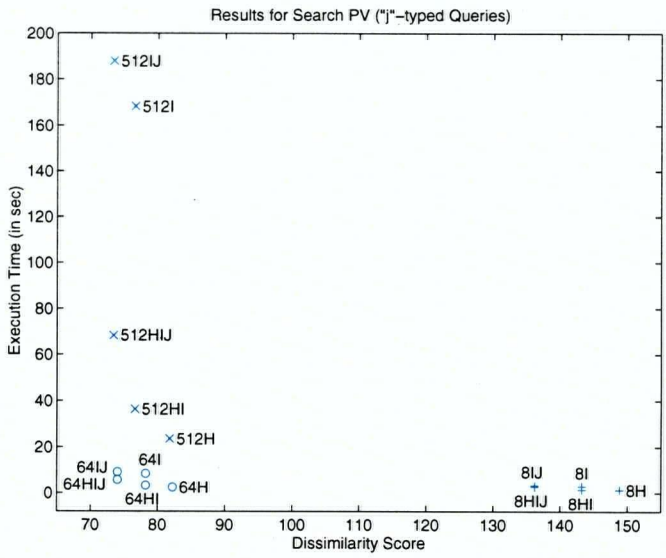


Figure 4.9: Experimental Results for Search PV ("j"-typed Queries)

To aim for a balance of efficiency and effectiveness, Scheme HIJ with 64-dimensional color histograms is the best filtering scheme for "j"-typed queries, when operated with Padding and Reduction Algorithms.

- "k"-typed queries

Filtering Scheme	Dimension of Color Histograms		
	8-dimensional	64-dimensional	512-dimensional
	η_A η_B η_C η_D	η_A η_B η_C η_D	η_A η_B η_C η_D
H	0.0 0.0 1.5 8.5	0.5 2.0 2.7 4.8	1.0 2.1 1.9 5.0
	$\Rightarrow DS = 160.0$	$\Rightarrow DS = 89.8$	$\Rightarrow DS = 89.6$
	$T_E = 1.2 \pm 0.0$ s	$T_E = 1.8 \pm 0.1$ s	$T_E = 18.9 \pm 6.6$ s
I	0.0 0.5 1.5 8.0	0.8 2.3 2.2 4.7	1.6 2.1 1.5 4.8
	$\Rightarrow DS = 150.0$	$\Rightarrow DS = 85.2$	$\Rightarrow DS = 82.8$
	$T_E = 1.6 \pm 0.2$ s	$T_E = 4.2 \pm 0.5$ s	$T_E = 124.1 \pm 11.1$ s
HI	0.0 0.5 1.5 8.0	0.8 2.3 2.2 4.7	1.6 2.1 1.5 4.8
	$\Rightarrow DS = 150.0$	$\Rightarrow DS = 85.2$	$\Rightarrow DS = 82.8$
	$T_E = 1.2 \pm 0.0$ s	$T_E = 2.0 \pm 0.1$ s	$T_E = 25.7 \pm 11.8$ s
IJ	0.4 0.3 1.3 8.0	1.3 2.0 2.1 4.6	1.6 2.2 1.5 4.7
	$\Rightarrow DS = 148.4$	$\Rightarrow DS = 81.8$	$\Rightarrow DS = 80.8$
	$T_E = 1.6 \pm 0.2$ s	$T_E = 4.4 \pm 0.7$ s	$T_E = 133.3 \pm 18.9$ s
HIJ	0.4 0.3 1.3 8.0	1.3 2.0 2.1 4.6	1.6 2.2 1.5 4.7
	$\Rightarrow DS = 148.4$	$\Rightarrow DS = 81.8$	$\Rightarrow DS = 80.8$
	$T_E = 1.2 \pm 0.1$ s	$T_E = 2.8 \pm 0.7$ s	$T_E = 52.8 \pm 34.3$ s
IJK	0.4 0.9 0.7 8.1	1.3 2.1 2.0 4.6	1.6 2.2 1.5 4.7
	$\Rightarrow DS = 147.5$	$\Rightarrow DS = 81.4$	$\Rightarrow DS = 80.8$
	$T_E = 1.8 \pm 0.4$ s	$T_E = 5.6 \pm 1.3$ s	$T_E = 167.7 \pm 52.6$ s
HIJK	0.4 0.9 0.7 8.1	1.3 2.1 2.0 4.6	1.6 2.2 1.5 4.7
	$\Rightarrow DS = 147.5$	$\Rightarrow DS = 81.4$	$\Rightarrow DS = 80.8$
	$T_E = 1.6 \pm 0.5$ s	$T_E = 4.4 \pm 1.5$ s	$T_E = 72.1 \pm 48.4$ s

Table 4.4: Experimental Results for Search PV ("k"-typed Queries)

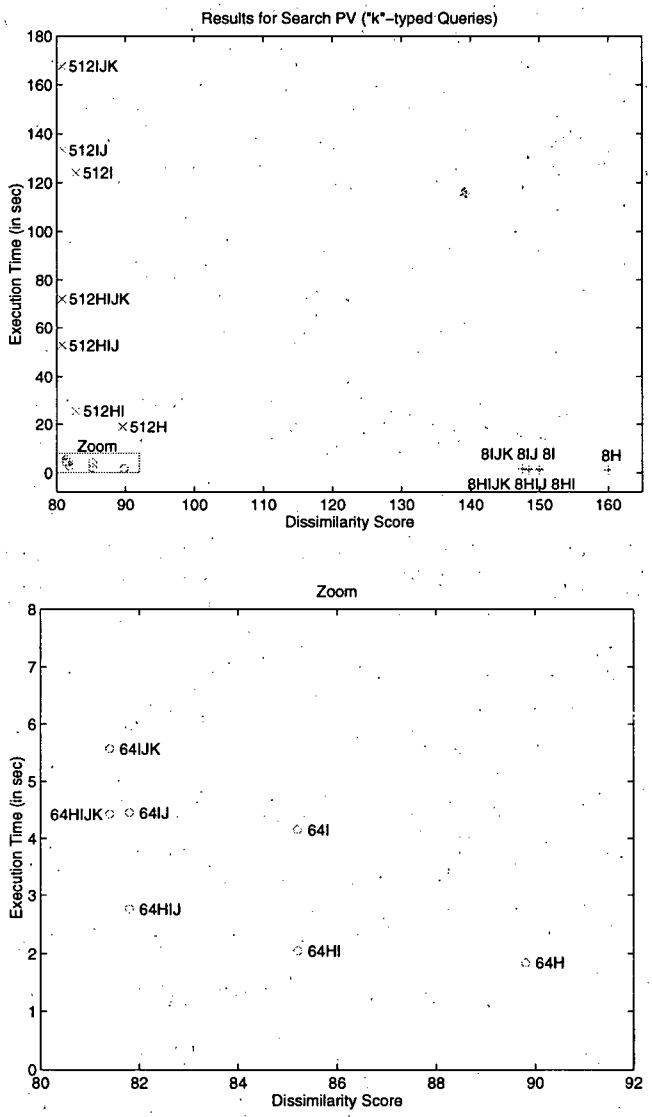


Figure 4.10: Experimental Results for Search PV ("k"-typed Queries)

To aim for a balance of performance and accuracy, the best filtering scheme for "k"-typed queries, as observed from the above experimental results, is Scheme HIJ with 64-dimensional color histograms.

4.4 Summary

Given that subimage queries can be of arbitrary size, picking one best scale at which the images are blocked is not easy. To cope with this challenge, a multiscale representation is proposed. In order to incorporate both color similarity and spatial similarity, the multiscale distance function is defined as a weighted sum of the histogram distance (D_H) and the positional distance (D_P):

$$D = \beta D_H + (1 - \beta) D_P$$

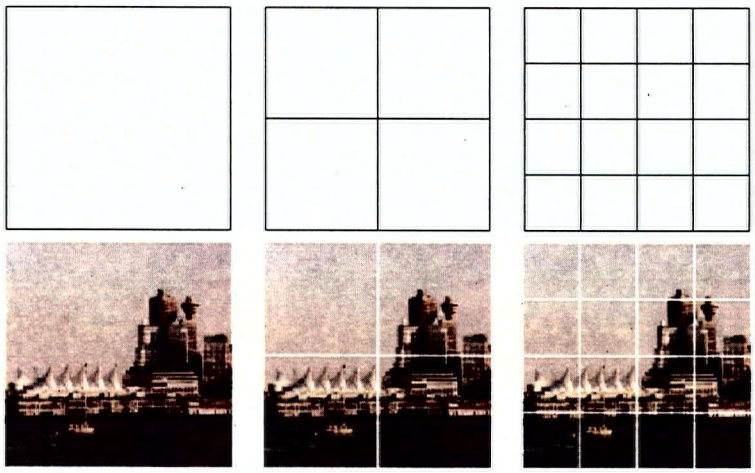
where the weighting factor (β) is a user preference for specifying the relative importance of the two distances above. The D_H can be estimated using Padding and Reduction Algorithms; the D_P can be estimated as the square of the shortest Euclidean distance between the query and the image subregion.

To aim for efficient and effective retrievals of desired images, the above multiscale distance function has also been formulated in such a way that given a query and an image subregion, the distance value at the coarser scale can serve as the distance value at the finer scale. With this formulation of the distance function, an efficient multiscale search strategy with the use of vertical filters (such as Search PV) is possible. Using Search PV, we analytically and experimentally investigated the suitable number of levels for the representation. In the analyses, we set up cost models to evaluate the 15 possible filtering schemes for the 4-level representation, and estimated their CPU, I/O, and combined CPU&I/O costs. The analytical results tend to favor the filtering schemes which start with Level H (or Level I), and those that do not skip intermediate levels. In the ex-

periments, subimage queries of arbitrary size were classified into four main types, and we measured the execution time (to evaluate the performance) and the dissimilarity score (to evaluate the degree of accuracy) for each filtering scheme that is favored by the analytical results. The experimental results show that when operated with Padding and Reduction Algorithms, Scheme H with 64-dimensional histograms is best for the "h"-typed queries, Scheme HI is best for the "i"-typed queries, and Scheme HIJ is best for both the "j"-typed and the "k"-typed queries. Moreover, among the three dimensions of histograms, while delivering better performance, the 8-dimensional cases suffer from a loss of accuracy. Conversely, while delivering better accuracy, the 512-dimensional cases suffer from a loss of performance/speed.

Based on these analytical and experimental results, we conclude that desired images can be retrieved efficiently and effectively using only the first three levels of the proposed representation. Hence, the recommended multiscale representation (Figure 4.11) can be described as follows:

- At Level H, the entire image is represented by a single global 64-dimensional color histogram.
- At Level I, the image is divided into four non-overlapping blocks, and each block is represented by a 64-dimensional color histogram covering $\frac{1}{4}$ of the entire image.
- At Level J, each block at Level I is further divided into four blocks, each of which is represented by a 64-dimensional color histogram covering $\frac{1}{16}$ of the entire image.



(a) Level H (b) Level I (c) Level J

Figure 4.11: The Recommended Multiscale Representation

Chapter 5

Search Strategies

To search a multiscale representation, some search strategies may require frequent jumping back and forth in the data file so as to get the necessary feature vectors representing the subregions of the images. However, such jumping makes it hard to optimize file organization and buffer management, and may generate a large number of I/Os. To avoid such jumping, a search strategy — Pure Vertical — was proposed and studied in Chapter 4. In addition to the Pure Vertical Search, we consider two other strategies, namely Pure Horizontal and Horizontal-and-Vertical. We investigate analytically and experimentally these strategies so as to find the best one which avoids frequent jumping and at the same time maintains a good balance of efficiency and effectiveness.

5.1 Problems of Branch-and-Bound Search

To use the branch-and-bound strategy for searching multiple scales [Chen *et al* 1997], all the database images are first checked at the coarsest scale, and the search proceeds to finer scales in non-descending order of distance value. In general, the branch-and-bound search works in such a way that it always keeps track of the distance values of all images contending for further consideration. Images with smallest values are "extended" to a finer scale. Then, these most recently "extended" images are considered along with the remaining ones. Again, images with smallest values are "extended". The process repeats until the target images are found.

Definition 5.1 To conduct the branch-and-bound search for retrieving the top u [1] images from the collection of M images:

1. Check all M images in the database at the coarse scale in the first iteration, and compute the distance value for each image.

2. Sort all images in non-descending order of distance value. Images with the current u smallest values become the current top-u.

3. Repeat until all the top u images have reached the finest scale:

   (a) For each of the u images,

      i. if the image has not reached the finest scale, extend one level/scale;

      ii. update the distance value of the extended image with the distance at the extended scale.

   (b) Sort all images in non-descending order of distance value, and obtain top u images for the next iteration. ■

[1] As mentioned in Chapter 4, u is the number of images requested by the user.

Due to the effectiveness of the brand-and-bound strategy for handling whole-image queries, this strategy can be adapted to handle subimage queries. However, at each iteration of the branch-and-bound search, the feature vectors used in the computation of distance values may be at a different level/scale and may be for images different from those in the previous iteration. As a result, it can be inefficient for large databases. In particular, the number of images the strategy must keep track of can be large. Jumping back and forth in the data file to get the necessary feature vectors for computation seems unavoidable.

Example 5.1 In many database applications, it is not unusual to retrieve the desired images from a collection of thousands of images. For simplicity, in this example, we try to find the top two images from a collection of seven images using the branch-and-bound search (with Scheme HIJ).

In the first iteration of the branch-and-bound search, all seven images are checked at the coarse scale (Level H), and a distance value is computed for each image. These images are then sorted in non-descending order of distance value, and the images with the two smallest values are the potential top two images. In each subsequent iteration, the potential top two images are extended one level/scale, and their distance values are updated; images are then sorted and a new set of the potential top two images is obtained.

In the following trace, the number at the slot representing an image at a particular level is the estimated distance value, and the superscript on its left denotes the search order (with the iteration number in brackets).

To find the top two images using the branch-and-bound search

	Img 1	Img 2	Img 3	Img 4	Img 5	Img 6	Img 7
Level H	$^{1st(1)}25$	$^{2nd(1)}5$	$^{3rd(1)}65$	$^{4th(1)}80$	$^{5th(1)}10$	$^{6th(1)}70$	$^{7th(1)}68$
Level I	$^{11th(3)}35$	$^{8th(2)}30$			$^{9th(2)}15$		
Level J		$^{12th(4)}32$			$^{10th(3)}20$		

■

Observe that jumping back and forth among images and levels may frequently be required, which makes it hard to optimize file organization and buffer management, and may impose a high I/O cost. Hence, we consider three search strategies — Search PV (Pure Vertical), Search PH (Pure Horizontal), and Search HV (Horizontal-and-Vertical) — which avoid such jumping.

5.2 Pure Vertical Search

We have studied Search PV (Pure Vertical) in the last chapter; here, we summarize the main points. Like the branch-and-bound search, Search PV is also accurate in its search. However, instead of jumping back and forth to get the necessary data during the search, Search PV checks the images one after another. At any point in time, a set of u images currently having smallest distance values are kept. For each image, the algorithm keeps proceeding to finer scales until (1) the distance value at a particular scale is already so large that the image cannot qualify as a good match, or (2) the finest scale is reached and the image is either discarded or selected as a member of the answer set, depending on the distance value.

To investigate the efficiency and the effectiveness of Search PV, analytical and experimental evaluations have been carried out. The results show that when operated with Padding and Reduction Algorithms, Scheme H with 64-dimensional histograms is best for the "h"-typed queries, Scheme HI is best for the "i"-typed queries, and Scheme HIJ is best for both the "j"-typed and the "k"-typed queries.

5.3 Pure Horizontal Search

Note that Search PV tends to require many comparisons at the finest scale, particularly at the beginning of the search. Thus, we consider another search strategy — Search PH

(Pure Horizontal) — that may require fewer comparisons at the finest scale. In Search PH, horizontal filters search "horizontally" across the database level by level, and the number of comparisons are predetermined. The idea is that all M images in the database are checked by a horizontal filter at the coarse scale in the first iteration. The best μ matches (where $u < \mu \ll M$, and μ is a predetermined parameter that controls the number of comparisons) are carried over from the current level to the next level (finer scale), while poor matches at the coarse scale are eliminated. In subsequent iterations, images left from the previous level are checked by horizontal filters at finer scales, and poor matches are again removed. The process repeats until the finest scale is reached and the top u images are returned.

Definition 5.2 (Search PH) To conduct the Pure Horizontal Search for retrieving the top u images from a collection of M images:

1. Check all M images at the coarsest scale/level, and compute the distance value for each image.
2. Sort the images in non-descending order of distance value. A subset of these images (the μ images with current smallest values) are carried over to the next level.
3. For each of the remaining levels:
   (a) Check the images which are carried over from the previous level, and compute the distance value for each image.
   (b) Sort the images in non-descending order of distance value; a subset of these images are again carried over to the next level.
   (c) Repeat Steps 3 (a) and (b) for the next level. ■

Example 5.2 Let us find the top two images from a collection of seven images using Search PH (with Scheme HIJ) [2]. In the following trace, the number at the slot representing an image at a particular level is the estimated distance value, and the superscript on its left denotes the search order.

To find the top two images using Search PH

	Img 1	Img 2	Img 3	Img 4	Img 5	Img 6	Img 7
Level H	^{1st} 25	^{2nd} 5	^{3rd} 65	^{4th} 80	^{5th} 10	^{6th} 70	^{7th} 68
	$\Rightarrow$ top five images are Imgs 1, 2, 3, 5, and 7						
Level I	^{8th} 35	^{9th} 30	^{10th} 200		^{11th} 15		^{12th} 78
	$\Rightarrow$ top three images are Imgs 1, 2, and 5						
Level J	^{13th} 67	^{14th} 32			^{15th} 20		
	$\Rightarrow$ top two images are Imgs 2 and 5						

■

Since the histograms are examined level by level, the best way to organize the precomputed feature vectors in the data file is to arrange them on a level-by-level basis. More precisely, the histograms of one level are followed by the histograms of another level (finer scale). Therefore, the file organization is of the form:

$$\underbrace{I^{\mathrm{H}}\, I^{\mathrm{H}} \cdots I^{\mathrm{H}}}_{\text{for } M \text{ images}} \quad \underbrace{I^{\mathrm{I}}\mathrm{s}\, I^{\mathrm{I}}\mathrm{s} \cdots I^{\mathrm{I}}\mathrm{s}}_{\text{for } M \text{ images}} \quad \underbrace{I^{\mathrm{J}}\mathrm{s}\, I^{\mathrm{J}}\mathrm{s} \cdots I^{\mathrm{J}}\mathrm{s}}_{\text{for } M \text{ images}} \quad \underbrace{I^{\mathrm{K}}\mathrm{s}\, I^{\mathrm{K}}\mathrm{s} \cdots I^{\mathrm{K}}\mathrm{s}}_{\text{for } M \text{ images}}$$

5.3.1 Analytical Evaluation

Cost Models

To measure the efficiency of Search PH, we set up cost models to estimate the CPU and the I/O costs. The CPU cost depends mainly on the time required to apply Padding and

[2] In many database applications, it is not unusual to retrieve the desired images from a collection of thousands of images. For simplicity, in the example, we try to retrieve two desired images from a collection of seven images.

Reduction Algorithms to the data (color histograms), and the computation time for each histogram (denoted by T_C) can be estimated using the experimental results in Section 3.4. The I/O cost depends mainly on the time required to sequentially or randomly access the pages containing the data (color histograms), and this access time can be affected by the size of data as well as the number of buffers. With the file organization described above, histograms at the first chosen level can be accessed sequentially. Given a minimum buffer size (one page) and an intelligent buffer management scheme, the number of random page accesses at the remaining levels can be estimated with the use of Cárdenas' Formula [Cárdenas 1975] or Yao's Formula [Yao 1977].

Definition 5.3 (Cárdenas' Formula) Given that R "records" divided into m pages and that r "records" satisfying the query are distributed uniformly among the m pages, the expression $1 - \frac{1}{m}$ gives the probability that a particular page does not contain a particular "record". If r "records" are selected independently, then the probability that a particular page not being hit is given by $(1 - \frac{1}{m})^r$, and $1 - (1 - \frac{1}{m})^r$ gives the probability that a particular page is hit. Therefore, the number of page accesses for a given query can be estimated using

$$C(m, r) = m \left[1 - \left(1 - \frac{1}{m}\right)^r\right] \tag{5.1}$$

■

Definition 5.4 (Yao's Formula) Given R "records" grouped into m pages (where $1 < m \leq R$), each contains $\frac{R}{m}$ "records". If r "records" (where $r \leq R - \frac{R}{m}$) are randomly selected from the R "records", the expected number of page accesses (pages with at least one "record" selected) is given by

$$Y(R, m, r) = m \left[1 - \prod_{j=1}^{r} \frac{R(1 - \frac{1}{m}) - j + 1}{R - j + 1}\right] \tag{5.2}$$

If $r > R - \frac{R}{m}$ or $m = 1$, then all m pages are expected to be accessed. ■

In the cost models, color histograms at each level for each image are treated as a "record". Depending on the level at which the histograms are represented, the size of "record" may vary. Sometimes a "record" may occupy a small portion of one page. For example, with a page size of 1 kilobyte (KB), the "record" of one Level-H 64-dimensional color histogram takes less than 30% of a page. With the file organization for Search PH, after the page containing a "record" of the Level-H histograms is loaded, no extra page access is needed for reading the Level-H "record" of the next image. However, sometimes a "record" may occupy more than one page. For example, with the above page size, the "record" of sixteen Level-J 64-dimensional color histograms occupies more than 15 pages. The cost of accessing all the histograms in this "record" is the sum of the time required to randomly access the first page and the time required to sequentially access the remaining pages.

Like Search PV, for the vast majority of subimage queries, in the case that the image subregions $\mathtt{I}^{\mathtt{sub}}$ and $\mathtt{I}^{\mathtt{sup}}$ are the best matches (which give the smallest $\widehat{D}$s) at their corresponding levels, $\mathtt{I}^{\mathtt{sub}}$ is one of the subregions enclosed in $\mathtt{I}^{\mathtt{sup}}$. When aiming for a balance of efficiency and effectiveness, checking all the histograms in a "record" may seem unnecessary, and only a portion of the "record" (some of the histograms) may need to be examined. As a result, both the CPU and the I/O costs can be reduced.

Example 5.3 In Scheme HIJ, we examine all M histograms at Level H, and carry the best μ_H images over to Level I where the 4 Level-I histograms for each of these images are examined. Then, at Level J, for each of the best μ_I images (where $\mu_H > \mu_I$) carried over from Level I, we examine the 4 histograms representing the subregions enclosed in the promising block of Level I. Hence, using a 64-dimensional color histogram with page size $P = 1$ KB, the cost model for Scheme HIJ can be described as follows:

- The CPU cost is the product of T_C and the total number of histograms used in the computation. In Scheme HIJ, Level-H histograms are examined for all M images,

4 Level-I histograms are checked for each of the μ_H images, and 4 Level-J histograms are examined for each of the μ_I images (where $M \gg \mu_H > \mu_I > u$, and both μ_H and μ_I are predefined). Hence, $CPU_{HIJ}^{PH} = (M + 4\mu_H + 4\mu_I)T_C$.

- The I/O cost is the sum of total seek times and total data transfer times. Using the above file organization for Search PH, an average seek time (denoted by T_A) is charged for moving the read head to the first "record" at Level H, and for each random page access of "records" at the remaining levels. When a page of data (color histograms) is loaded, a data transfer time (denoted by T_D) is charged.

  More precisely, for Scheme HIJ, T_A is needed to get to the Level-H histogram of the first image, and T_D is charged for loading the page containing the histogram. Since all Level-H histograms are stored contiguously, no seek is charged for accessing subsequent Level-H histograms, and only T_D is charged for each sequential page access of data. At Levels I and J, the number of random page accesses can be estimated by Cárdenas' Formula, and each random page access requires one T_A and one T_D. Hence, $I/O_{HIJ}^{PH} = [1 + C(M, \mu_H) + C(4M, \mu_I)]T_A + \left[\frac{M}{4} + C(M, \mu_H) + C(4M, \mu_I)\right]T_D$.

The cost models for other filtering schemes, color histograms of other dimensions, other page sizes, or for Yao's Formula can be formulated in a similar manner. ■

The cost models for the 15 filtering schemes (with page size $P = 1$ KB, using 64-dimensional color histograms and Cárdenas' Formula) are shown below:

1. Scheme H

   We examine all M histograms at Level H.

   - $CPU_H^{PH} = MT_C$
   - $I/O_H^{PH} = T_A + \frac{M}{4}T_D$

2. Scheme I

   We examine all $4M$ histograms at Level I.

   - $CPU_I^{PH} = 4MT_C$
   - $I/O_I^{PH} = T_A + MT_D$

3. Scheme HI

   We examine all M histograms at Level H, and carry the best μ_H images over to Level I where the 4 histograms for each of these images are examined.

   - $CPU_{HI}^{PH} = (M + 4\mu_H)T_C$
   - $I/O_{HI}^{PH} = [1 + C(M, \mu_H)]T_A + \left[\frac{M}{4} + C(M, \mu_H)\right]T_D$

   where μ_H is a predefined parameter that controls the number of images to be carried over from Level H to the next level (Level I) [3], and its value lies between u and M.

4. Scheme J

   We examine all $16M$ histograms at Level J.

   - $CPU_J^{PH} = 16MT_C$
   - $I/O_J^{PH} = T_A + 4MT_D$

5. Scheme HJ

   We examine all M histograms at Level H, and carry the best μ_H images over to Level J where the 16 histograms for each of these images are examined.

   - $CPU_{HJ}^{PH} = (M + 16\mu_H)T_C$
   - $I/O_{HJ}^{PH} = [1 + C(4M, \mu_H)]T_A + \left[\frac{M}{4} + 4C(4M, \mu_H)\right]T_D$

[3] Similarly, μ_I and μ_J in the following cost models control the number of images to be carried over to the next available level from Level I and Level J respectively.

6. Scheme IJ

   We examine all $4M$ histograms at Level I, and carry the best μ_I images over to Level J where for each of these images, the 4 histograms representing the subregions enclosed in the promising block of Level I are examined.

   - $CPU_{IJ}^{PH} = (4M + 4\mu_I)T_C$
   - $I/O_{IJ}^{PH} = [1 + C(4M, \mu_I)]\,T_A + [M + C(4M, \mu_I)]\,T_D$

7. Scheme HIJ

   We examine all M histograms at Level H, and carry the best μ_H images over to Level I where the 4 Level-I histograms for each of these images are examined. Then, at Level J, for each of the best μ_I images (where $\mu_H > \mu_I$) carried over from Level I, we examine the 4 histograms representing the subregions enclosed in the promising block of Level I.

   - $CPU_{HIJ}^{PH} = (M + 4\mu_H + 4\mu_I)T_C$
   - $I/O_{HIJ}^{PH} = [1 + C(M, \mu_H) + C(4M, \mu_I)]\,T_A + \left[\frac{M}{4} + C(M, \mu_H) + C(4M, \mu_I)\right] T_D$

8. Scheme K

   We examine all $64M$ histograms at Level K.

   - $CPU_K^{PH} = 64MT_C$
   - $I/O_K^{PH} = T_A + 16MT_D$

9. Scheme HK

   We examine all M histograms at Level H, and carry the best μ_H images over to the Level K where the 64 histograms for each of these images are examined.

   - $CPU_{HK}^{PH} = (M + 64\mu_H)T_C$
   - $I/O_{HK}^{PH} = [1 + C(16M, \mu_H)]\,T_A + \left[\frac{M}{4} + 16C(16M, \mu_H)\right] T_D$

10. Scheme IK

We examine all $4M$ histograms at Level I, and carry the best μ_I images over to Level K where for each of these images, the 16 histograms representing the subregions enclosed in the promising block of Level I are examined.

- $CPU_{IK}^{PH} = (4M + 16\mu_I)T_C$
- $I/O_{IK}^{PH} = [1 + C(16M, \mu_I)]\,T_A + [M + 4C(16M, \mu_I)]\,T_D$

11. Scheme JK

We examine all $16M$ histograms at Level J, and carry the best μ_J images over to Level K where for each of these images, the 4 histograms representing the subregions enclosed in the promising block of Level J are examined.

- $CPU_{JK}^{PH} = (16M + 4\mu_J)T_C$
- $I/O_{JK}^{PH} = [1 + C(16M, \mu_J)]\,T_A + [4M + C(16M, \mu_J)]\,T_D$

12. Scheme HIK

We examine all M histograms at Level H, and carry the best μ_H images over to Level I where the 4 Level-I histograms for each of these images are examined. Then, at Level K, for each of the best μ_I images (where $\mu_H > \mu_I$) carried over from Level I, we examine the 16 histograms representing the subregions enclosed in the promising block of Level I.

- $CPU_{HIK}^{PH} = (M + 4\mu_H + 16\mu_I)T_C$
- $I/O_{HIK}^{PH} = [1 + C(M, \mu_H) + C(16M, \mu_I)]\,T_A + \left[\frac{M}{4} + C(M, \mu_H) + 4C(16M, \mu_I)\right] T_D$

13. Scheme HJK

We examine all M histograms at Level H, and carry the best μ_H images over to Level J where the 16 Level-J histograms for each of these images are examined.

Then, at Level K, for each of the best μ_J images (where $\mu_H > \mu_J$) carried over from Level J, we examine the 4 histograms representing the subregions enclosed in the promising block of Level J.

- $CPU^{PH}_{HJK} = (M + 16\mu_H + 4\mu_J)T_C$
- $I/O^{PH}_{HJK} = [1 + C(4M, \mu_H) + C(16M, \mu_J)]\,T_A + \left[\frac{M}{4} + 4C(4M, \mu_H) + C(16M, \mu_J)\right] T_D$

14. Scheme IJK

We examine all $4M$ histograms at Level I, and carry the best μ_I images over to Level J where for each of these images, the 4 histograms representing the subregions enclosed in the promising block of Level I are examined. Then, at Level K, for each of the best μ_J images (where $\mu_I > \mu_J$) carried over from Level J, we examine the 4 histograms representing the subregions enclosed in the promising block of Level J.

- $CPU^{PH}_{IJK} = (4M + 4\mu_I + 4\mu_J)T_C$
- $I/O^{PH}_{IJK} = [1 + C(4M, \mu_I) + C(16M, \mu_J)]\,T_A + [M + C(4M, \mu_I) + C(16M, \mu_J)]\,T_D$

15. Scheme HIJK

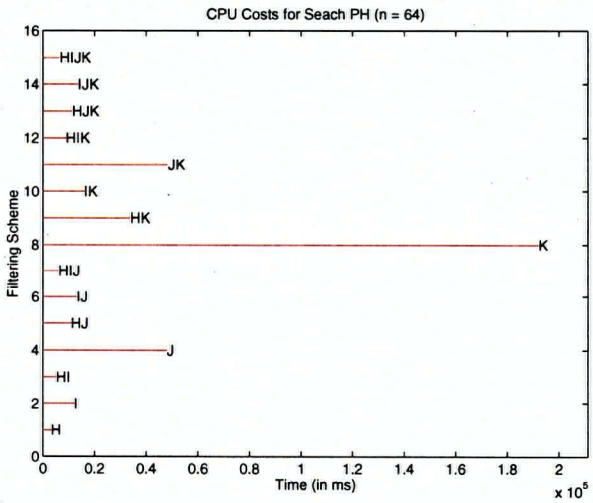
We examine all M histograms at Level H, and carry the best μ_H images over to Level I where the 4 Level-I histograms for each of these images are examined. Then, at Level J, for each of the best μ_I images (where $\mu_H > \mu_I$) carried over from Level I, we examine the 4 histograms representing the subregions enclosed in the promising block of Level I, and carry the best μ_J images (where $\mu_I > \mu_J$) over to Level K where the 4 Level-K histograms for each of these images are examined.

- $CPU^{PH}_{HIJK} = (M + 4\mu_H + 4\mu_I + 4\mu_J)T_C$

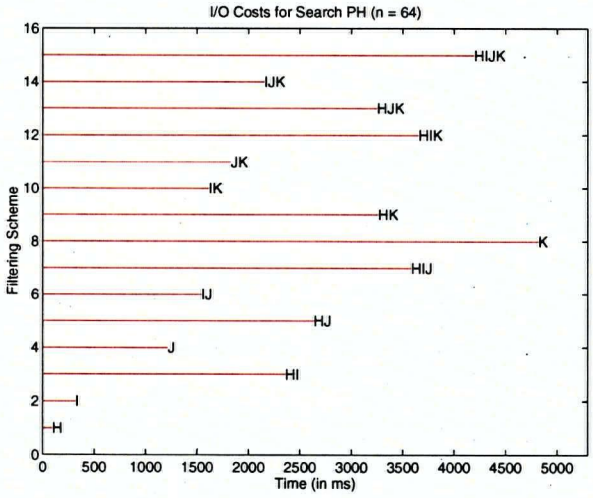
- $I/O^{PH}_{HIJK} = [1 + C(M, \mu_H) + C(4M, \mu_I) + C(16M, \mu_J)]\, T_A$
  $+ \left[\frac{M}{4} + C(M, \mu_H) + C(4M, \mu_I) + C(16M, \mu_J)\right] T_D$

Analytical Results

The above cost models provide a good foundation for analyzing the 15 filtering schemes, and for discovering general trends. The computation time for Padding and Reduction Algorithms (T_C) depends on the sizes of the subimage query and the image subregion. For the analysis, we let T_C be 3 ms for the 8-dimensional and the 64-dimensional histograms, and 20 ms for the 512-dimensional histograms. We assume that each minimum seek (T_M) takes 5 ms, each average seek (T_A) requires 15 ms, and the transfer of data in each page (T_D) with page size $P = 1$ KB needs 0.3 ms, and that μ_H, μ_I and μ_J are 160, 80, and 40 respectively. Analyses on finding $u = 10$ images from a database of M=1000 images were then conducted. As observed from the experimental results of Search PV, while delivering better performance, the filtering schemes with the 8-dimensional histogram suffer from a loss of effectiveness. By contrast, while delivering better accuracy, the schemes with the 512-dimensional histogram suffer from a loss of efficiency. So, we summarize only the analytical results for the 64-dimensional color histograms in the figures on the next few pages. In the figures, for each filtering scheme, the CPU, I/O, or combined CPU&I/O cost is shown by the solid line.

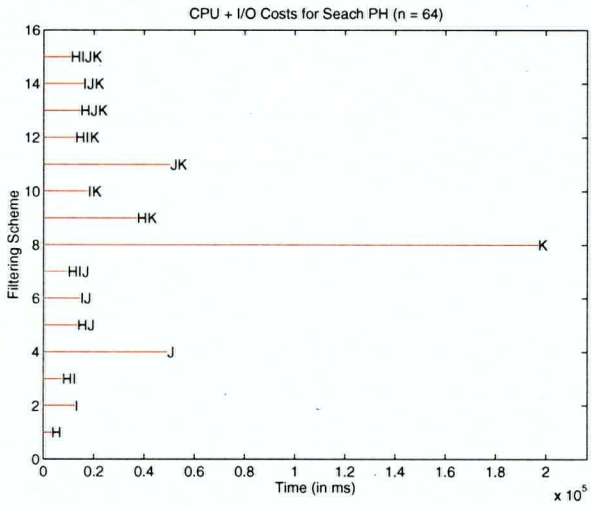


(a) CPU Costs for Search PH



(b) I/O Costs for Search PH

Figure 5.1: Analytical Results for Search PH (64-dimensional Color Histograms)



(c) Combined CPU and I/O Costs for Search PH

Figure 5.1: Analytical Results for Search PH (Continued)

Observation 5.1 The CPU and the I/O costs for every filtering scheme are deterministic. The best-case analyses are the same as the worst-case analyses.

Explanation Since the number of images to be carried over from the current level to the next level is controlled by the predefined parameter μ_j, we know statically the number of images to be considered and to be retained at a particular level. ■

As for Search PV, the following trends are observed:

1. The CPU cost is affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required for CPU operations increases.

2. The I/O cost is affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required for I/O operations increases.

3. The combined CPU and I/O cost is affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required for both CPU and I/O operations increases.

4. Filtering schemes which start with filter J or K take more CPU and I/O time. For example, the combined CPU and I/O times demanded by Schemes J, JK, and K are longer than the times taken by the worst cases of many filtering schemes such as Schemes HIJ and HIJK.

5. Filtering schemes which skip intermediate levels incur greater CPU and I/O cost.

6. The combined CPU and I/O cost for the additional intermediate levels becomes relatively less expensive as the dimension of the histogram grows. For example, as the dimension of the histogram increases, the combined cost for Scheme HJ is more expensive than that for Scheme HIJ. Similarly, the combined cost for Scheme HK is more expensive than that for Scheme HIK or HJK, and either of these two costs is more expensive than that for Scheme HIJK.

In general, the above analytical results tend to favor the filtering schemes which start with Level H (or Level I), and those that do not skip any intermediate level. More precisely, Schemes H, I, HI, IJ, HIJ, IJK, and HIJK are favored.

5.3.2 Experimental Evaluation

To find the appropriate filtering schemes for Search PH, several experiments have been performed using the same set of data (color histograms) as for the experimental evaluation of Search PV. Again, for each subimage query, we categorize the database images into five main classes, and the effectiveness of the search strategy is assessed by counting the number of retrieved images which fall into each class. The dissimilarity score (DS) for the retrieval of the best 10 images is computed as:

$$DS = 7(2 - \eta_A) + 5(3 - \eta_B) + (5 - \eta_C) + 15\eta_D + 40\eta_E$$

where η_j indicates the number of images falling into Class j. The efficiency of Search PH can be assessed using the execution time (T_E) which measures the combined CPU&I/O

cost per query on a database of 1000 images.

Let μ_H, μ_I and μ_J be 160, 80, and 40 respectively. The experiments were run on a Sun UltraSPARC-1 workstation using a page size of 1 KB and user preference β of 0.5 [4]. The results (the average DS and T_E values for the retrievals of the best 10 images from a collection of 1000 images) are summarized in the tables and figures on the next few pages. The T_E measures the efficiency and the DS assesses the effectiveness.

- "h"-typed queries

Filtering Scheme	64-dimensional Color Histograms					
	η_A	η_B	η_C	η_D	$\Rightarrow DS$	T_E
H	1.9	2.1	1.7	4.3	73.0	3.4 $\pm$ 1.9 s

Table 5.1: Experimental Results for Search PH ("h"-typed Queries)

- "i"-typed queries

Filtering Scheme	64-dimensional Color Histograms					
	η_A	η_B	η_C	η_D	$\Rightarrow DS$	T_E
H	1.5	2.2	1.8	4.5	78.2	4.1 $\pm$ 1.6 s
I	1.7	2.1	1.8	4.4	75.8	5.9 $\pm$ 4.3 s
HI	1.7	2.1	1.8	4.4	75.8	5.3 $\pm$ 1.1 s

Table 5.2: Experimental Results for Search PH ("i"-typed Queries)

[4] With the user preference β set to 0.5, both the color distribution and spatial information are of the same level of importance.

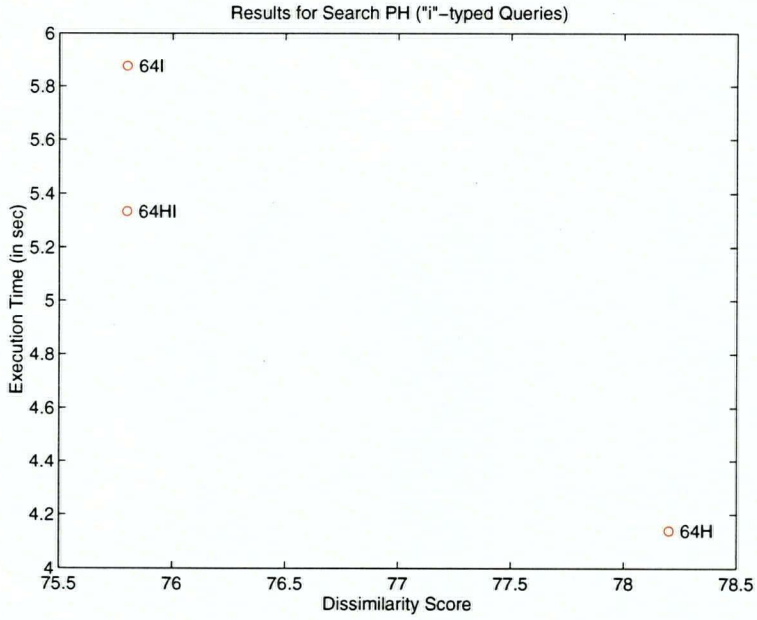


Figure 5.2: Experimental Results for Search PH ("i"-typed Queries)

- "j"-typed queries

Filtering Scheme	64-dimensional Color Histograms					
	η_A	η_B	η_C	η_D	$\Rightarrow DS$	T_E
H	0.9	2.1	2.5	4.5	82.2	1.4 ± 0.2 s
I	1.5	2.2	1.8	4.5	78.2	7.4 ± 1.6 s
HI	1.5	2.2	1.8	4.5	78.2	2.2 ± 0.3 s
IJ	1.6	2.0	2.1	4.3	75.2	7.7 ± 1.6 s
HIJ	1.6	2.0	2.1	4.3	75.2	2.5 ± 0.3 s

Table 5.3: Experimental Results for Search PH ("j"-typed Queries)

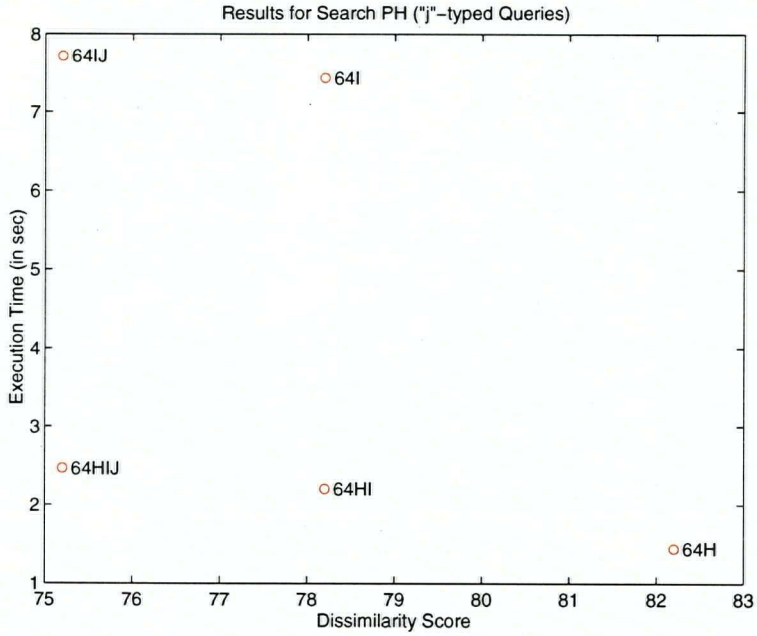


Figure 5.3: Experimental Results for Search PH ("j"-typed Queries)

- "k"-typed queries

Filtering Scheme	64-dimensional Color Histograms					
	η_A	η_B	η_C	η_D	$\Rightarrow DS$	T_E
H	0.5	2.0	2.7	4.8	89.8	0.7 ± 0.1 s
I	0.8	2.3	2.2	4.7	85.2	3.1 ± 0.5 s
HI	0.8	2.3	2.2	4.7	85.2	1.3 ± 0.1 s
IJ	1.1	2.1	2.2	4.6	82.6	3.3 ± 0.5 s
HIJ	1.1	2.1	2.2	4.6	82.6	1.4 ± 0.2 s
IJK	1.2	2.0	2.2	4.6	82.4	3.4 ± 0.5 s
HIJK	1.2	2.0	2.2	4.6	82.4	1.6 ± 0.2 s

Table 5.4: Experimental Results for Search PH ("k"-typed Queries)

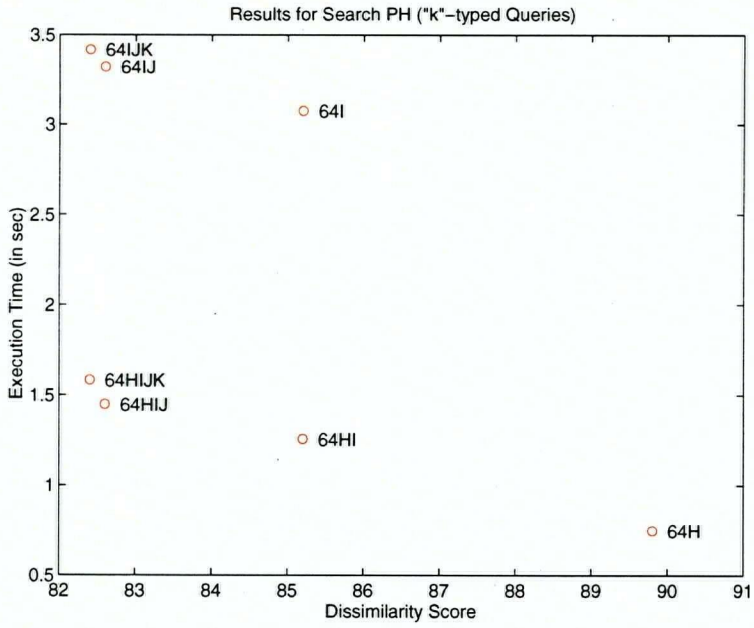


Figure 5.4: Experimental Results for Search PH ("k"-typed Queries)

Like Search PV, when operated with Padding and Reduction Algorithms, Scheme H is best for"h"-typed queries, Scheme HI is best for "i"-typed queries, and Scheme HIJ is best for both "j"-typed and "k"-typed queries. Unlike Search PV, the DS values for Search PH appear to be higher than those for Search PV. However, the T_E values for Search PH appear to be smaller than those for Search PV. In other words, while delivering efficiency, Search PH suffers from a loss of effectiveness.

5.4 Horizontal-and-Vertical Search

Note that in Search PH, horizontal filters are used not only at one level, but at all the levels. Hence, the number of images to be carried over from one level to another needs to be chosen carefully. If this set of numbers (μ_H, μ_I, and μ_J) is not determined carefully (say, the numbers are too small), then for some queries, an image `I` that gives a good match at a finer scale could have been eliminated before reaching this finer scale. This

may happen when there are sufficiently many images which are not as good as I at the finer scale but which are better than I at the coarser scale. Consequently, while delivering efficiency, Search PH may suffer from a loss of effectiveness. So, we consider another strategy — Search HV (Horizontal-and-Vertical) — which is a hybrid of Search PV and Search PH. With it, a horizontal filter is applied only to the coarsest level [5], and vertical filters are then applied to the remaining levels. The idea is that all M images in the database are checked by a horizontal filter at the coarse scale in the first iteration. The best μ matches (where $u < \mu \ll M$) are carried over from the current level to the next level (finer scale), while poor matches at the coarse scale are eliminated. Then, all these best μ images are checked one after another using vertical filters. At any point in time, the current u smallest distances (at the finest scale of the u images, where $u < \mu$) are kept. When an image is tested, if its distance value at the current scale is already greater than some distance value of the current u smallest, the tested image can be eliminated. Otherwise, a finer scale is used. The process repeats until the image is (1) eliminated or (2) added to become one of the current u smallest (when it reaches the finest scale). By so doing, the detailed search with the use of vertical filters is applied not to the set of all the images, but only to its most promising subset.

Definition 5.5 (Search HV) To conduct the Horizontal-and-Vertical Search for retrieving the top u images from a collection of M images:

1. Check all M images at the coarsest scale/level, and compute the distance value for each image.

2. Sort the images in non-descending order of distance value. A subset of these images (the μ images with current smallest values) are carried over to the next level.

[5] With one level of horizontal level, we only need to carefully assign a value to one (instead of three) predefined parameter μ that controls the images to be carried over.

3. The first u images in this subset are checked at all remaining levels, and the distance value is computed for each image at the finest scale. These u images become the current top-u.

4. For each of the remaining $\mu - u$ images:

   (a) Let the second coarsest scale be the current level/scale.

   (b) Compute the distance value. If the value is less than some value of the top-u, extend one level/scale (provided that we have not reached the finest level) and repeat Step 4 (b) for the extended scale. ■

Example 5.4 Let us find the top two images from a collection of seven images using Search HV (with Scheme HIJ). In the following trace, the number at the slot representing an image at a particular level is the estimated distance value, and the superscript on its left denotes the search order.

To find the top two images using Search HV

	Img 1	Img 2	Img 3	Img 4	Img 5	Img 6	Img 7
Level H	^{1st} 25	^{2nd} 5	^{3rd} 65	^{4th} 80	^{5th} 10	^{6th} 70	^{7th} 68
	$\Rightarrow$ top five images are Images 1, 2, 3, 5, and 7						
Level I	^{8th} 35	^{10th} 30	^{12th} 200		^{13th} 15		^{15th} 78
Level J	^{9th} 67	^{11th} 32			^{14th} 20		
	$\Downarrow$	$\Downarrow$	$\Downarrow$		$\Downarrow$		$\Downarrow$
Top two images	Img 1	Imgs 2 and 1	Imgs 2 and 1		Imgs 5 and 2		Imgs 5 and 2

■

In Search HV, the histograms are examined image by image at the coarsest scale, and level by level at the remaining scales. So, the way in which the precomputed feature vectors

in the data file are organized depends on the filtering scheme. More precisely, for the schemes which started at Level H, the best file organization is of the form:

$$\underbrace{I^{H}\, I^{H} \cdots I^{H}}_{\text{for } M \text{ images}} \quad \underbrace{I^{I}{}_{S}\, I^{J}{}_{S}\, I^{K}{}_{S}}_{\text{for Image 1}} \quad \underbrace{I^{I}{}_{S}\, I^{J}{}_{S}\, I^{K}{}_{S}}_{\text{for Image 2}} \quad \cdots \quad \underbrace{I^{I}{}_{S}\, I^{J}{}_{S}\, I^{K}{}_{S}}_{\text{for Image } M}$$

Similarly, for the schemes which started at Level I, the best file organization is of the form:

$$\underbrace{I^{I}\, I^{I} \cdots I^{I}}_{\text{for } M \text{ images}} \quad \underbrace{I^{J}{}_{S}\, I^{K}{}_{S}}_{\text{for Image 1}} \quad \underbrace{I^{J}{}_{S}\, I^{K}{}_{S}}_{\text{for Image 2}} \quad \cdots \quad \underbrace{I^{J}{}_{S}\, I^{K}{}_{S}}_{\text{for Image } M}$$

and for the schemes which started at Levels J and K, the best file organization is of the form:

$$\underbrace{I^{J}{}_{S}\, I^{J}{}_{S} \cdots I^{J}{}_{S}}_{\text{for } M \text{ images}} \quad \underbrace{I^{K}{}_{S}\, I^{K}{}_{S} \cdots I^{K}{}_{S}}_{\text{for } M \text{ images}}$$

In general, the histograms of the coarsest scale are arranged in such a way that the histograms of one image are followed by those of another image. By so doing, an efficient sequential read of these histograms at the coarsest scale is possible. For the remaining scales, histograms are arranged in such a way that for each image, the histograms at the second coarsest scale are followed by the histograms at finer scales.

5.4.1 Analytical Evaluation

Cost Models

To measure the efficiency of Search HV, we set up cost models to estimate the CPU and the I/O costs. The CPU cost depends mainly on the number of histograms to be examined and the computation time T_C; the I/O cost depends mainly on the time required to sequentially or randomly access the pages containing the data (color histograms). With the file organization described above, histograms at the first chosen level can be accessed sequentially. Given a minimum buffer size (one page) and an intelligent buffer management scheme, the number of page accesses at the remaining levels can be estimated with the use of statistical formulations.

Example 5.5 In Scheme HIJ, we examine all M histograms at Level H. Then, for each of the best μ_H images carried over from Level H, we examine the 4 Level-I histograms, and if necessary [6], we consider the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I. Hence, using a 64-dimensional color histogram with page size $P = 1$ KB, the cost model for Scheme HIJ can be described as follows:

- The CPU cost is the product of T_C and the total number of histograms used in the computation. In Scheme HIJ, Level-H histograms are examined for all M images, 4 Level-I histograms are checked for each of the μ_H images, and 4 Level-J histograms are examined for each of the ξ_5 images (where $M \gg \mu_H \geq \xi_5 \geq u$, and μ_H is predefined but ξ_5 is determined dynamically at runtime). Hence, $CPU^{HV}_{HIJ} = (M + 4\mu_H + 4\xi_5)T_C$.

- The I/O cost is the sum of total seek times and total data transfer times. Using the appropriate file organization for Search HV mentioned above, an average seek time (denoted by T_A) is charged for moving the read head to the first "record" at Level H, and for each random page access of "records" at the second level (Level I). Since all Level-I and Level-J "records" for a particular image are stored contiguously, only a minimum seek time (denoted by T_M) is charged for each jump between the data (for example, jumping from the fourth Level-I histogram to the third Level-J histogram of the same image). Moreover, when a page of data (color histograms) is loaded, a data transfer time (denoted by T_D) is charged.

  More precisely, for Scheme HIJ, T_A is needed to get to the Level-H histogram of the first image, and T_D is charged for loading the page containing the histogram. Since all Level-H histograms are stored contiguously, no seek is charged for accessing subsequent Level-H histograms, and only T_D is charged for each sequential page

[6] We consider the 4 Level-J histograms of an image if the $\widehat{D}$ at Level I is less than any $\widehat{D}$ of the current top-u.

access of data. Then, T_A is needed to get to the Level-I histograms of each image carried over from Level H, and T_D is charged for loading the page containing the histograms. Since only 4 histograms can fit into a page, if the 4 Level-J histograms happen to be examined, then we load the next page. There is a probability of $\frac{1}{4}$ that the selected 4 Level-J histograms are stored contiguously after the loaded Level-I histograms. Hence, $I/O_{HIJ}^{HV} = (1+\mu_H)T_A + \frac{3\xi_5}{4}T_M + (\frac{M}{4} + \mu_H + \xi_5)T_D$.

The cost models for other filtering schemes, color histograms of other dimensions, or other page sizes can be formulated in a similar manner. ■

The cost models for the 15 filtering schemes (with page size $P = 1$ KB and using 64-dimensional color histograms) are shown below:

1. Scheme H

   We examine all M histograms at Level H.

   - $CPU_H^{HV} = MT_C$
   - $I/O_H^{HV} = T_A + \frac{M}{4}T_D$

2. Scheme I

   We examine all $4M$ histograms at Level I.

   - $CPU_I^{HV} = 4MT_C$
   - $I/O_I^{HV} = T_A + MT_D$

3. Scheme HI

   We examine all M histograms at Level H. For each of the best μ_H images carried over from Level H, we examine the 4 Level-I histograms.

   - $CPU_{HI}^{HV} = (M + 4\mu_H)T_C$

- $I/O_{HI}^{HV} = (1+\mu_H)T_A + (\frac{M}{4}+\mu_H)T_D$

where μ_H is a predefined parameter that controls the number of images to be carried over from Level H to the next level (Level I) [7], and its value lies between u and M.

4. Scheme J

We examine all $16M$ histograms at Level J.

- $CPU_J^{HV} = 16MT_C$
- $I/O_J^{HV} = T_A + 4MT_D$

5. Scheme HJ

We examine all M histograms at Level H. For each of the best μ_H images carried over from Level H, we examine the 16 Level-J histograms.

- $CPU_{HJ}^{HV} = (M + 16\mu_H)T_C$
- $I/O_{HJ}^{HV} = (1+\mu_H)T_A + (\frac{M}{4}+4\mu_H)T_D$

6. Scheme IJ

We examine all $4M$ histograms at Level I. For each of the best μ_I images carried over from Level I, we examine the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I.

- $CPU_{IJ}^{HV} = (4M + 4\mu_I)T_C$
- $I/O_{IJ}^{HV} = (1+\mu_I)T_A + (M+\mu_I)T_D$

7. Scheme HIJ

We examine all M histograms at Level H. For each of the best μ_H images carried over from Level H, we examine the 4 Level-I histograms, and if necessary [8], consider

[7] Similarly, μ_I and μ_J in the following cost models control the number of images to be carried over to the next available level from Level I and Level J respectively.

[8] We consider the 4 Level-J histograms of an image if the $\widehat{D}$ at Level I is less than any $\widehat{D}$ of the current top-u. Similar conditions apply to the filtering schemes involving more than two levels.

the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I.

- $CPU_{HIJ}^{HV} = (M + 4\mu_H + 4\xi_5)T_C$
- $I/O_{HIJ}^{HV} = (1 + \mu_H)T_A + \frac{3\xi_5}{4}T_M + (\frac{M}{4} + \mu_H + \xi_5)T_D$

where $\mu_H \geq \xi_5 \geq u$.

8. Scheme K

   We examine all $64M$ histograms at Level K.

   - $CPU_K^{HV} = 64MT_C$
   - $I/O_K^{HV} = T_A + 16MT_D$

9. Scheme HK

   We examine all M histograms at Level H. For each of the best μ_H images carried over from Level H, we examine the 64 Level-K histograms.

   - $CPU_{HK}^{HV} = (M + 64\mu_H)T_C$
   - $I/O_{HK}^{HV} = (1 + \mu_H)T_A + (\frac{M}{4} + 16\mu_H)T_D$

10. Scheme IK

    We examine all $4M$ histograms at Level I. For each of the best μ_I images carried over from Level I, we examine the 16 Level-K histograms.

    - $CPU_{IK}^{HV} = (4M + 16\mu_I)T_C$
    - $I/O_{IK}^{HV} = (1 + \mu_I)T_A + (M + 4\mu_I)T_D$

11. Scheme JK

    We examine all $16M$ histograms at Level J. For each of the best μ_J images carried over from Level J, we examine the 4 Level-K histograms.

- $CPU_{JK}^{HV} = (16M + 4\mu_J)T_C$
- $I/O_{JK}^{HV} = (1 + \mu_J)T_A + (4M + \mu_J)T_D$

12. Scheme HIK

We examine all M histograms at Level H. For each of the best μ_H images carried over from Level H, we examine the 4 Level-I histograms, and if necessary, consider the 16 Level-K histograms which represent the subregions enclosed in the promising block of Level I.

- $CPU_{HIK}^{HV} = (M + 4\mu_H + 16\xi_{10})T_C$
- $I/O_{HIK}^{HV} = (1 + \mu_H)T_A + \xi_{10}T_M + (\frac{M}{4} + \mu_H + 4\xi_{10})T_D$

where $\mu_H \geq \xi_{10} \geq u$.

13. Scheme HJK

We examine all M histograms at Level H. For each of the best μ_H images carried over from Level H, we examine the 16 Level-J histograms, and if necessary, consider the 4 Level-K histograms which represent the subregions enclosed in the promising block of Level J.

- $CPU_{HJK}^{HV} = (M + 16\mu_H + 4\xi_{12})T_C$
- $I/O_{HJK}^{HV} = (1 + \mu_H)T_A + \frac{15\xi_{12}}{16}T_M + (\frac{M}{4} + 4\mu_H + \xi_{12})T_D$

where $\mu_H \geq \xi_{12} \geq u$.

14. Scheme IJK

We examine all $4M$ histograms at Level I. For each of the best μ_I images carried over from Level I, we examine the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I, and if necessary, consider the 4 Level-K histograms which represent the subregions enclosed in the promising block of Level J.

- $CPU^{HV}_{IJK} = (4M + 4\mu_I + 4\xi_{14})T_C$
- $I/O^{HV}_{IJK} = (1 + \mu_I)T_A + \frac{15\xi_{14}}{16}T_M + (M + \mu_I + \xi_{14})T_D$

where $\mu_I \geq \xi_{14} \geq u$.

15. Scheme HIJK

We examine all M histograms at Level H. For each of the best μ_H images carried over from Level H, we examine the 4 Level-I histograms, then consider (if necessary) the 4 Level-J histograms which represent the subregions enclosed in the promising block of Level I, and consider (if necessary) the 4 Level-K histograms which represent the subregions enclosed in the promising block of Level J.

- $CPU^{HV}_{HIJK} = (M + 4\mu_H + 4\xi_{16} + 4\xi_{17})T_C$
- $I/O^{HV}_{HIJK} = (1 + \mu_H)T_A + (\frac{3\xi_{16}}{4} + \xi_{17})T_M + (\frac{M}{4} + \mu_H + \xi_{16} + \xi_{17})T_D$

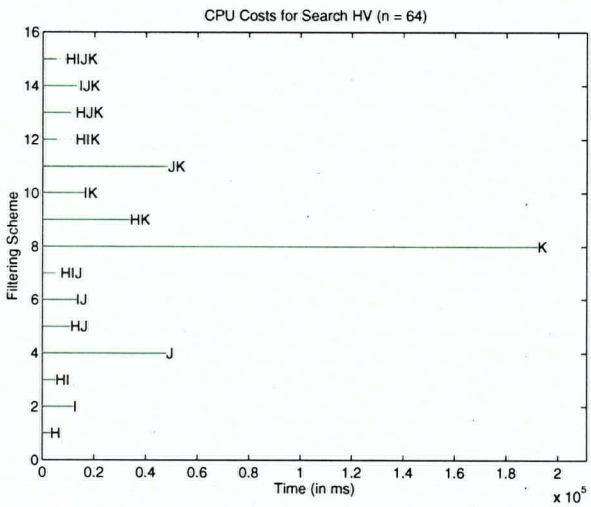
where $\mu_H \geq \xi_{16} \geq \xi_{17} \geq u$.

Note from the cost models for filtering schemes involving more than two levels, the value of each ξ_j is determined dynamically. For instance, in the best case of Scheme HIJ, the first u images carried over from Level H are the top-u, and each Level-I distance for the remaining $\mu_H - u$ images is greater than all Level-J distances of the top u images. As such, the histograms at Level J for the $\mu_H - u$ images are not examined; thus, $\xi_5 = u$. Conversely, in the worst case, at any point in time the Level-I distance of each image is less than some Level-J distance of the current top-u. As such, all histograms at all levels for the μ_H images are examined; thus, $\xi_5 = \mu_H$.

Analytical Results

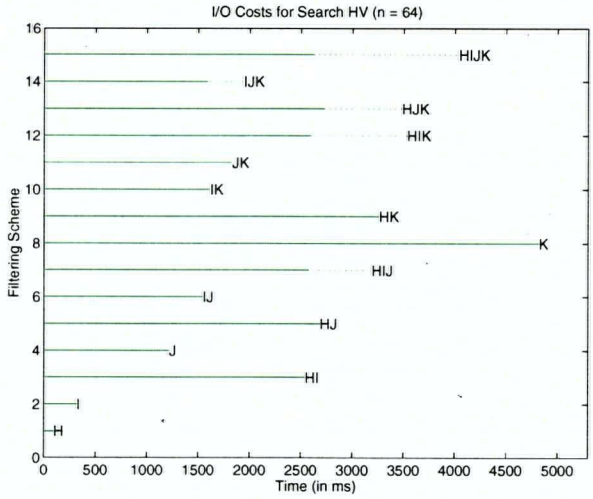
The above cost models provide a good foundation for analyzing the 15 filtering schemes, and for discovering general trends. The computation time for Padding and Reduction

Algorithms (T_C) depends on the sizes of the subimage query and the image subregion. For the analysis, we let T_C be 3 ms for the 8-dimensional and the 64-dimensional histograms, and 20 ms for the 512-dimensional histograms. We assume that each minimum seek (T_M) takes 5 ms, each average seek (T_A) requires 15 ms, and the transfer of data in each page (T_D) with page size P = 1 KB needs 0.3 ms, and that μ_H, μ_I and μ_J are 160, 80, and 40 respectively. Analyses on finding u = 10 images from a database of M=1000 images were then conducted. As in Search PH, we summarize only the analytical results for the 64-dimensional color histograms in the figures on the next few pages. In the figures, for each filtering scheme, the minimum cost is shown by the solid line, and the additional cost (if any) is indicated by the dotted line.

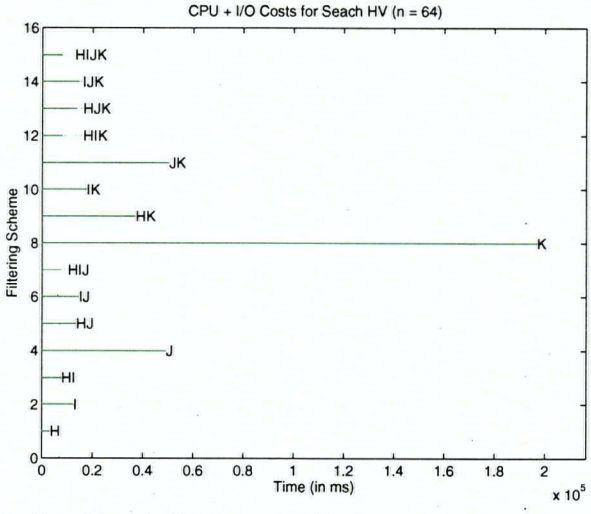


(a) CPU Costs for Search HV

Figure 5.5: Analytical Results for Search HV (64-dimensional Color Histograms)



(b) I/O Costs for Search HV



(c) Combined CPU and I/O Costs for Search HV

Figure 5.5: Analytical Results for Search HV (Continued)

As for Searches PV and PH, the following trends are observed:

1. The CPU cost is affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required for CPU operations increases.

2. The I/O cost is affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required for I/O operations increases.

3. The combined CPU and I/O cost is affected by the dimension of the color histogram: As the dimension of the histogram grows, the time required for both CPU and I/O operations increases.

4. Filtering schemes which start with filter J or K often take more CPU and I/O time. For example, the combined CPU and I/O times demanded by Schemes J, JK, and K are often longer than the times taken by the worst cases of many filtering schemes such as Schemes HIJ and HIJK.

5. Filtering schemes which skip intermediate levels often incur greater CPU and I/O cost.

6. The combined CPU and I/O cost for the additional intermediate levels becomes relatively less expensive as the dimension of the histogram grows. For example, as the dimension of the histogram increases, the combined cost for Scheme HJ is usually more expensive than that for Scheme HIJ. Similarly, the combined cost for Scheme HK is usually more expensive than that for Scheme HIK or HJK, and either of these two costs is more expensive than that for Scheme HIJK.

In general, the above analytical results tend to favor the filtering schemes which start with Level H (or Level I), and those that do not skip any intermediate level. More precisely, Schemes H, I, HI, IJ, HIJ, IJK, and HIJK are favored.

5.4.2 Experimental Evaluation

To find the appropriate filtering schemes for Search HV, several experiments have been performed using the same set of data (color histograms) as for the experimental evaluation of Search PV. Again, for each subimage query, we categorize the database images into five main classes, and the effectiveness of the search strategy is assessed by counting the number of retrieved images which fall into each class. The dissimilarity score (DS) for the retrieval of the best 10 images is computed as:

$$DS = 7(2 - \eta_A) + 5(3 - \eta_B) + (5 - \eta_C) + 15\eta_D + 40\eta_E$$

where η_j indicates the number of images falling into Class j. The efficiency of Search HV can be assessed using the execution time (T_E) which measures the combined CPU&I/O cost per query on a database of 1000 images.

Let μ_H, μ_I and μ_J be 160, 80, and 40 respectively. The experiments were run on a Sun UltraSPARC-1 workstation using a page size of 1 KB and user preference β of 0.5. The results (the average DS and T_E values for the retrievals of the best 10 images from a collection of 1000 images) are summarized in the tables and figures on the next few pages. The T_E measures the efficiency and the DS assesses the effectiveness.

- "h"-typed queries

Filtering Scheme	64-dimensional Color Histograms					
	η_A	η_B	η_C	η_D	$\Rightarrow DS$	T_E
H	1.9	2.1	1.7	4.3	73.0	3.6 $\pm$ 1.9 s

Table 5.5: Experimental Results for Search HV ("h"-typed Queries)

- "i"-typed queries

Filtering Scheme	64-dimensional Color Histograms					
	η_A	η_B	η_C	η_D	$\Rightarrow DS$	T_E
H	1.5	2.2	1.8	4.5	78.2	4.4 ± 1.6 s
I	1.7	2.1	1.8	4.4	75.8	8.4 ± 4.2 s
HI	1.7	2.1	1.8	4.4	75.8	6.3 ± 1.3 s

Table 5.6: Experimental Results for Search HV ("i"-typed Queries)

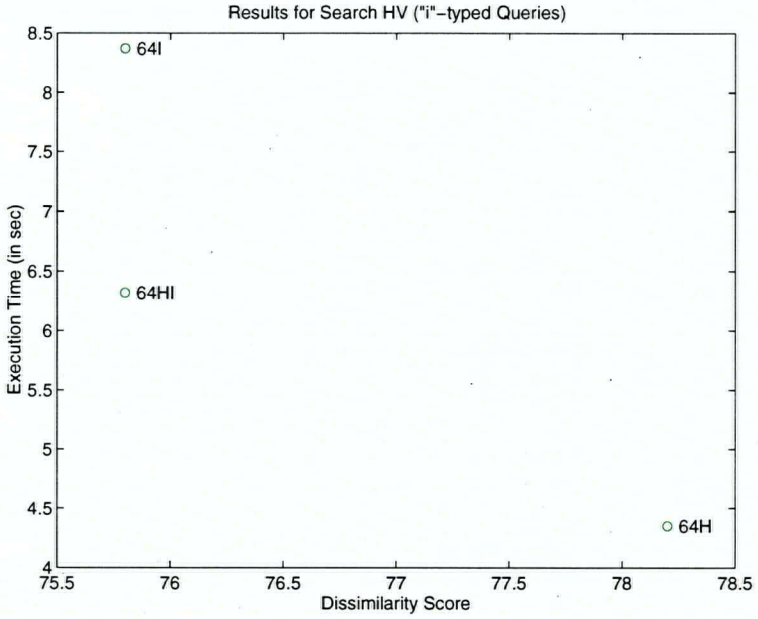


Figure 5.6: Experimental Results for Search HV ("i"-typed Queries)

- "j"-typed queries

Filtering Scheme	64-dimensional Color Histograms					
	η_A	η_B	η_C	η_D	$\Rightarrow DS$	T_E
H	0.9	2.1	2.5	4.5	82.2	1.7 ± 0.2 s
I	1.5	2.2	1.8	4.5	78.2	7.7 ± 1.6 s
HI	1.5	2.2	1.8	4.5	78.2	2.5 ± 0.4 s
IJ	1.8	2.0	1.9	4.3	74.0	8.0 ± 1.6 s
HIJ	1.8	2.0	1.9	4.3	74.0	2.8 ± 0.3 s

Table 5.7: Experimental Results for Search HV ("j"-typed Queries)

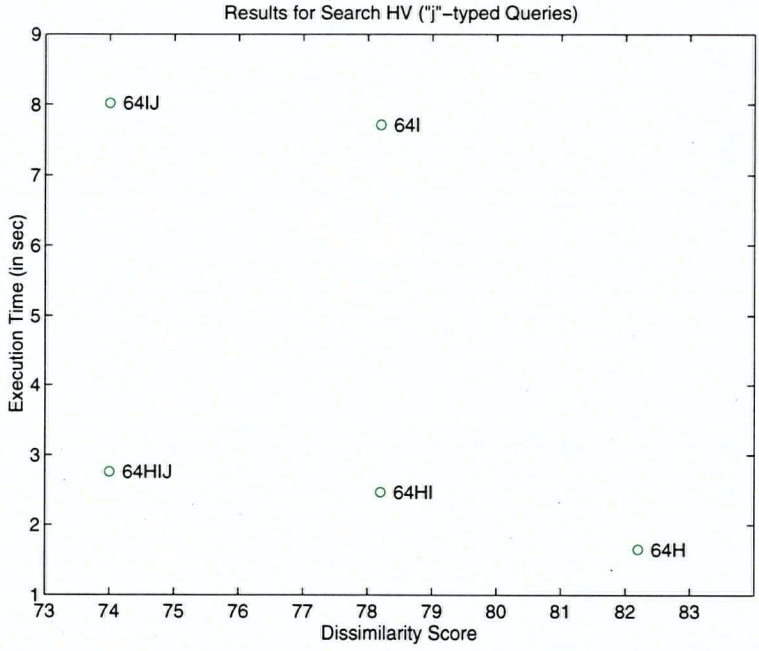


Figure 5.7: Experimental Results for Search HV ("j"-typed Queries)

- "k"-typed queries

Filtering Scheme	64-dimensional Color Histograms					
	η_A	η_B	η_C	η_D	$\Rightarrow DS$	T_E
H	0.5	2.0	2.7	4.8	89.8	0.9 ± 0.1 s
I	0.8	2.3	2.2	4.7	85.2	3.4 ± 0.5 s
HI	0.8	2.3	2.2	4.7	85.2	1.7 ± 0.2 s
IJ	1.2	2.1	2.1	4.6	82.0	3.6 ± 0.5 s
HIJ	1.2	2.1	2.1	4.6	82.0	1.5 ± 0.2 s
IJK	1.3	2.1	2.0	4.6	81.4	3.8 ± 0.5 s
HIJK	1.3	2.1	2.0	4.6	81.4	2.0 ± 0.2 s

Table 5.8: Experimental Results for Search HV ("k"-typed Queries)

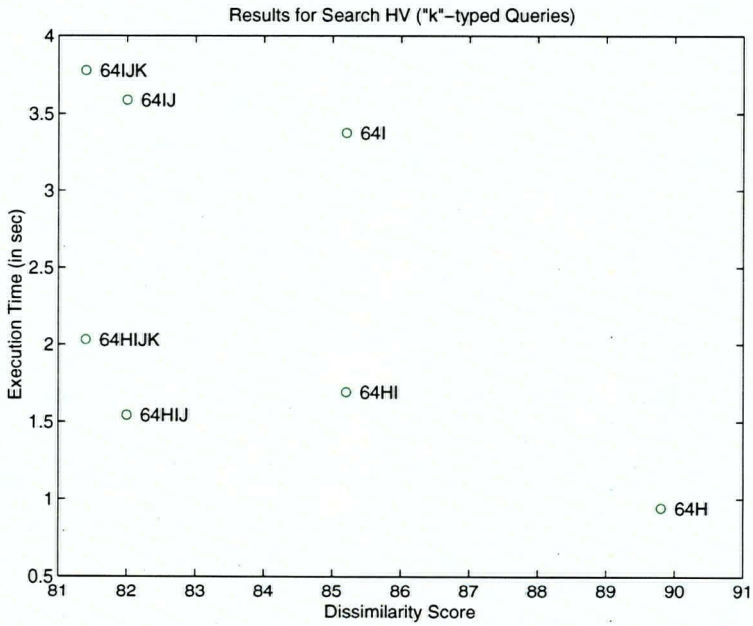


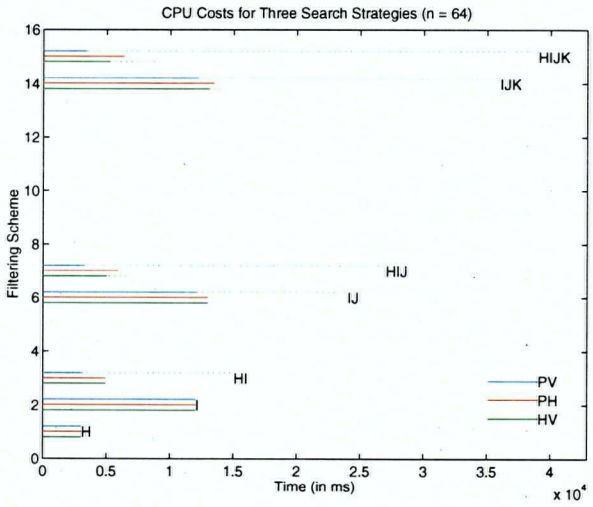
Figure 5.8: Experimental Results for Search HV ("k"-typed Queries)

Like Searches PV and PH, when operated with Padding and Reduction Algorithms, Scheme H is best for"h"-typed queries, Scheme HI is best for "i"-typed queries, and Scheme HIJ is best for both "j"-typed and "k"-typed queries. Unlike Search PH, the DS values for Search HV appear to be lower than those for Search PH; unlike Search PV, the T_E values for Search HV appear to be smaller than those for Search PV. In other words,

Search HV keeps a good balance of efficiency and effectiveness.

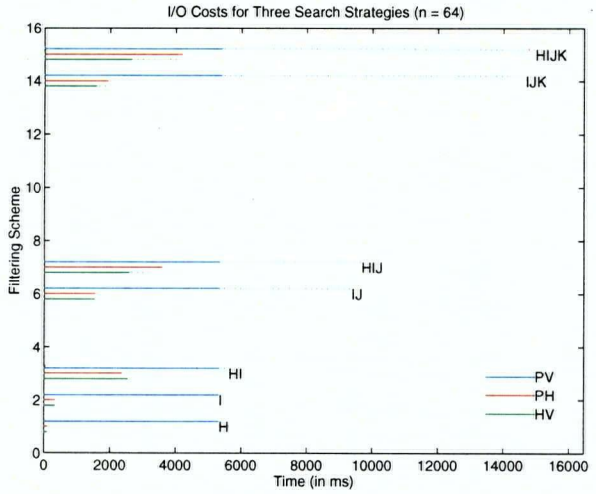
5.5 The Best Search Strategy

Three search strategies — namely, Search PV, Search PH, and Search HV — have been suggested to avoid the kind of frequent jumping incurred in the branch-and-bound strategy. To find the best strategy among the three, analyses and experiments have been carried out. Analytically, we set up cost models to evaluate the filtering schemes and estimate their CPU, I/O, and combined CPU&I/O costs. All three strategies share a common trend: Filtering schemes starting with Level H (or Level I) and those not skipping intermediate levels are considered to be favorable.

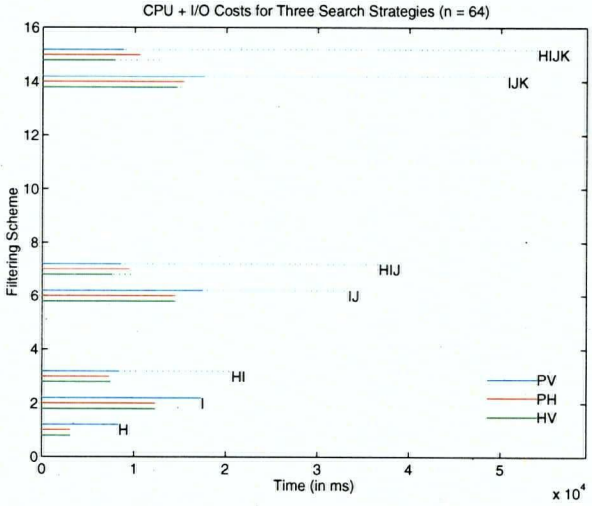


(a) CPU Costs for Three Search Strategies

Figure 5.9: Analytical Results for Three Search Strategies (64-dimensional Color Histograms)



(b) I/O Costs for Three Search Strategies



(c) Combined CPU and I/O Costs for Three Search Strategies

Figure 5.9: Analytical Results for Three Search Strategies (Continued)

As observed from Figure 5.9, the CPU, I/O, and combined CPU&I/O costs for Search PV are generally higher than those for the other two strategies. The costs for Searches PH and HV are usually almost the same.

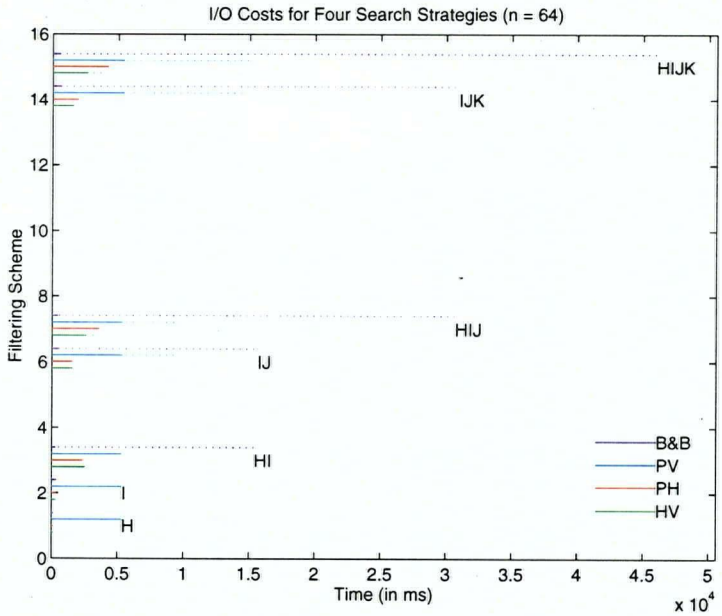


Figure 5.10: I/O Costs for Four Search Strategies (64-dimensional Color Histograms)

As observed from the above figure, the I/O costs for the branch-and-bound strategy (denoted by B&B and indicated by purple lines) are potentially high. This explains why we need to consider the three search strategies which reduce the I/O costs.

Experimentally, subimage queries of arbitrary size are classified into four main types, and we measure the execution time (showing the performance) and the dissimilarity score (showing the degree of accuracy) for each of these favorite schemes. Again, all three strategies agree on the results obtained when operated with Padding and Reduction Algorithms. Scheme H with 64-dimensional histograms is the best filtering scheme for "h"-typed queries, Scheme HI is the best one for "i"-typed queries, and Scheme HIJ is the best for both "j"-typed and "k"-typed queries.

Query Type	Search Strategies		
	Search PV	Search PH	Search HV
	η_A η_B η_C η_D	η_A η_B η_C η_D	η_A η_B η_C η_D
"h"	1.9 2.1 1.7 4.3	1.9 2.1 1.7 4.3	1.9 2.1 1.7 4.3
Scheme	$\Rightarrow DS = 73.0$	$\Rightarrow DS = 73.0$	$\Rightarrow DS = 73.0$
H	$T_E = 4.45 \pm 3.80$ s	$T_E = 3.39 \pm 1.90$ s	$T_E = 3.56 \pm 1.91$ s
"i"	1.8 2.1 1.8 4.3	1.7 2.1 1.8 4.4	1.7 2.1 1.8 4.4
Scheme	$\Rightarrow DS = 73.6$	$\Rightarrow DS = 75.8$	$\Rightarrow DS = 75.8$
HI	$T_E = 7.15 \pm 0.76$ s	$T_E = 5.33 \pm 1.11$ s	$T_E = 6.32 \pm 1.33$ s
"j"	1.8 2.0 1.9 4.3	1.6 2.0 2.1 4.3	1.8 2.0 1.9 4.3
Scheme	$\Rightarrow DS = 74.0$	$\Rightarrow DS = 75.2$	$\Rightarrow DS = 74.0$
HIJ	$T_E = 5.80 \pm 1.66$ s	$T_E = 2.47 \pm 0.28$ s	$T_E = 2.76 \pm 0.33$ s
"k"	1.3 2.0 2.1 4.6	1.1 2.1 2.2 4.6	1.2 2.1 2.1 4.6
Scheme	$\Rightarrow DS = 81.8$	$\Rightarrow DS = 82.6$	$\Rightarrow DS = 82.0$
HIJ	$T_E = 2.76 \pm 0.73$ s	$T_E = 1.45 \pm 0.16$ s	$T_E = 1.54 \pm 0.21$ s

Table 5.9: Experimental Results for Three Search Strategies

Comparing the runtimes and the accuracies of the best configuration for each strategy, we found that Search HV is more efficient than Search PV, because the use of vertical filters in the latter is applied to the whole set of all M images. In Search HV, the detailed search with the use of vertical filters is applied not to the set of all the images, but only to its most promising subset.

Search HV is more effective than Search PH, because the latter uses horizontal filters at all levels. If the number of images to be carried over from the current level to the next level is not determined carefully (say, the number is too small), then for some queries, an image `I` that gives a good match at a finer scale could have been eliminated before reaching this finer scale. This may happen when there are sufficiently many images which are not as good as `I` at the finer scale, but better than `I` at the coarser scale. Consequently, while delivering efficiency, Search PH may suffer from a loss of effectiveness. In Search HV, a horizontal filter is applied not to all the levels, but to the coarsest level. So, we only need to carefully assign a value to one (instead of three) predefined parameter μ that

controls the number of images to be carried over. As such, the chance of having the above mentioned I being eliminated before reaching the finer scale is reduced.

Moreover, the experimental results on Searches PV, PH, and HV suggest that Search HV is the best strategy for the retrieval of desired images when operated with Padding and Reduction Algorithms. The reason is that in addition to avoiding the kind of frequent jumping observed in the branch-and-bound strategy, Search HV also keeps a balance of efficiency and effectiveness.

5.6 Summary

Although the branch-and-bound algorithm is accurate in its search, it can be inefficient for large databases. In particular, the number of images the algorithm must keep track of can be large, and jumping back and forth among images and scales may frequently be required. To avoid such jumping, we studied three strategies, namely Search PV, Search PH, and Search HV.

In Search PV, the images are checked one after another using vertical filters. At any point in time, a set of u images currently having smallest distance values are kept. For each image, the algorithm keeps proceeding to finer scales until (1) the distance value at a particular scale is already so large that the image cannot be qualified as a good match, or (2) the finest scale is reached and the image is either discarded or selected as a member of the answer set, depending on the distance value.

In Search PH, all the images in the database are checked by the horizontal filter at the coarsest scale in the first iteration. Poor matches at this scale are then eliminated. In subsequent iterations, images that are left from the previous scale/level are checked by filters at finer scales, and poor matches are again removed. The process repeats until the finest scale is reached and the top u images are returned.

Search HV is a hybrid of the above two in which we use a horizontal filter on the

first level and vertical filters on the remaining levels. The results of the analytical and experimental investigations show that Search HV is the best strategy for avoiding the frequent jumping and at the same time keeping a good balance of performance/speed and accuracy. When operated with Padding and Reduction Algorithms, the best 10 desired images can be retrieved efficiently and effectively from a collection of a thousand images in about 3.5 seconds.

Chapter 6

Conclusions

6.1 Conclusions

As network connectivity has continued its explosive growth and storage devices have become smaller, faster, and less expensive, the number of on-line digital images has increased rapidly. We can no longer rely solely on traditional database retrieval technology based on manually associating textual descriptions with image contents. The development of efficient and effective content-based retrieval systems based on automated extracted contents (such as color) are necessary. In order to achieve this goal, several research projects have been carried out. Over the past few years, multi-dimensional indexing structures have been designed, multi-level filtering approaches have been proposed, and information preserving transformations have been suggested for providing efficient indexing. Expressive query language systems have also been developed to accommodate efficient querying from image databases for applications with dense and sparse image spaces. Other research projects for supporting efficient and effective query processing and optimization have also been carried out.

With the popularity of similarity matching on color and spatial information, many image database management systems store the information in local color histograms. To

process user queries, some systems use fixed grid segmentation approaches. For further improvement on the efficiency and the effectiveness of content-based retrieval, multiscale matching approaches have been proposed. However, detailed analytical and experimental results on the determination of the suitable number of levels for these approaches are seldom reported, and comparisons of different strategies for searching multiple scales are also rare. Moreover, in many situations, users are interested in, or can remember, only local image contents; therefore, subimage query processing is needed. Unfortunately, not many image database management systems can handle arbitrary-size subimage queries based on color and spatial similarity. For the systems that can deal with subimage queries of arbitrary size, multiscale matching is rarely used.

In this thesis, our first issue of investigation was to find a method for dealing with arbitrary-size subimage queries. To answer queries of this kind, some systems segment an image into several blocks, each of which has an associated color histogram. One problem with this arrangement is that subimage queries may be of arbitrary size that not necessarily an integral multiple of the chosen block size. Other systems use template-based matching algorithms. A key problem with those algorithms is that a lot of computation is needed, because of the large number of positions to be compared. Correspondingly, we proposed two algorithms, called Padding and Reduction, for dealing with subimage queries of arbitrary size. Knowing the image subregion $\mathtt{I}$ represented by the precomputed feature vector may not necessarily contain the same number of pixels as the query $\mathtt{Q}$, we used Padding and Reduction Algorithms to estimate the best possible color histograms for $\mathtt{Q}$ and $\mathtt{I}$. Here, we either (1) enlarged $\mathtt{Q}$ into a new query $\mathtt{Q}'$ that is of the same size as $\mathtt{I}$ or (2) reduced $\mathtt{I}$ to a new image subregion $\mathtt{I}'$ that is of the same size as $\mathtt{Q}$. In terms of effectiveness, both algorithms give the same best-case lower bound to the histogram distance. In terms of efficiency, the Padding Algorithm outperforms the Reduction Algorithm when the size differential between the subimage query and the image subregion is

large, and *vice versa* when the differential is small.

Given subimage queries of arbitrary size, multiscale representation may improve the efficiency and the effectiveness of content-based retrievals. The idea is that depending on the scale or the need of a given query, a more appropriate scale can be used. So, a 4-level representation was proposed such that the entire image is divided into four subregions, and each subregion is recursively divided into four subregions, and so on. Since image contents are usually pre-extracted and stored, our second issue of investigation was to determine the suitable number of levels for such a representation. To do so, we analytically and experimentally studied the performance and the accuracy of the multi-level filtering schemes available for the representation. In the analyses, we used cost models to estimate the CPU, I/O, and combined CPU and I/O costs for each scheme. The results favor the schemes which start with the coarsest level (or the second coarsest level) of the 4-level representation and those that do not skip intermediate levels. In the experiments, given a subimage query, we measured the execution time (showing the performance) and computed the dissimilarity score (showing the accuracy). The results suggest that when using Padding and Reduction Algorithms, the desired images can be retrieved efficiently and effectively using only the top three levels. Hence, a 3-level hierarchy (up to the 4×4 segmentation) is preferred.

Several strategies for searching multiple scales can be applied to the retrieval of desired images from a collection of database images which are stored in the multiscale representation. Branch-and-bound is one of the strategies; it is accurate in its search, but it can be inefficient for large databases. In particular, the number of images to be kept track of can be quite large, and frequent jumping back and forth among the data may be required. So, our third issue of investigation was to find an efficient and effective strategy for searching multiple scales. In this thesis, we studied three search strategies, namely Pure Vertical, Pure Horizontal, and Horizontal-and-Vertical. In the Pure Vertical

Search, images are checked "vertically" by vertical filters one after another, and a set of potentially desired images is kept. After a new image is checked, it is either discarded or selected as a member of the answer set. In the Pure Horizontal Search, images are checked "horizontally" level by level using horizontal filters. After images are checked at a particular level, poor matches at that level are eliminated, and all remaining images are carried over to the next level where they are checked by another horizontal filter. The hybrid strategy — Horizontal-and-Vertical — uses a horizontal filter on the first level and vertical filters on the remaining levels for the most promising subset of the database images. To find the best strategy among the three, analytical and experimental evaluations were performed. The results indicate that the Horizontal-and-Vertical Search is the best when operated with Padding and Reduction Algorithms, not only because the Search avoids the kind of frequent jumping as in the branch-and-bound, but because it keeps a good balance of performance/speed and accuracy. With it, the best 10 desired images can be retrieved efficiently and effectively from a collection of a thousand images in about 3.5 seconds.

6.2 Future Work

Although the results of the thesis are very promising, there are some aspects to consider for further improvement. One aspect is to investigate methods to extend our Padding and Reduction Algorithms for dealing with arbitrary-size subimage queries of arbitrary shapes. For instance, a user may want to find images with a ship in the bottom right corner. It is well known that human memory is weak in retaining a fine granularity of spatial information of color. The user may not remember (or he may not be interested in) any other portion of the images (not even a region next to the ship). In such a case, the subimage query he wants to submit may be of irregular shape (as shown in Figure 6.1, for example). Our Padding and Reduction Algorithms may be able to estimate the histogram distance. However, the algorithm for estimating the positional distance between the query

and the image subregion may need to be modified; computational geometry techniques may be required for handling queries of complex irregular shapes.

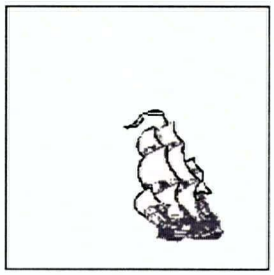


Figure 6.1: Example of a Subimage Query of Arbitrary Shape

Again, in some situations, due to the poor human memory capability for retaining a fine granularity of spatial information of color, the user may only be able to recall the information on the boundary of the region he is interested in, but not the center. In such cases, the query is no longer "solid" but "hollow". Examples of these queries are a ring-like query and the one in Figure 6.2. If the "hole" is small, Padding and Reduction Algorithms are expected to provide a reasonable estimation to the histogram distance. However, if the "hole" turns out to be large, care needs to be taken in handling this "hollow" subimage query of arbitrary size. Moreover, in some other situations, the user may not care about the color information on the "hole", and it can match any color. To deal with these kind of queries containing "don't care" parts, approaches similar to those for handling "hollow" queries can be applied.

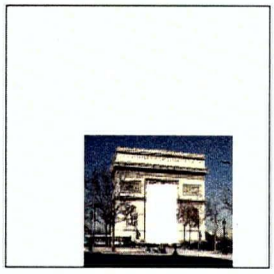


Figure 6.2: Example of a "Hollow" Subimage Query

Since some users have poor memories, they may demand support for "hollow"

queries and queries containing "don't care" parts. However, other users may remember more than one portion of the images they have seen before. Hence, methods for handling subimage queries with these multiple "known" portions (for example, Figure 6.3) are also needed. A naïve approach is to apply Padding and Reduction Algorithms to each portion independently. An intersection is then applied to the candidate sets of images returned by the Algorithms, and the resulting images are ranked thereafter. One problem with this naïve approach is that for a subimage query with multiple "known" portions, performing Padding and Reduction for each portion may be time-consuming. Hence, more efficient approaches are necessary for handling arbitrary-size subimage queries of arbitrary shapes, "hollow" queries, queries containing "don't care" parts, and queries with multiple "known" portions.

Figure 6.3: Example of a Subimage Query with Multiple "Known" Portions

Enhancements to multiscale search strategy should also be explored. In the process of determining the appropriate number of levels in the multiscale representation and finding the best search strategy, the histograms which are examined during a search are of the same dimension. For example, with the best scheme for handling "j"-typed queries, all the histograms checked at Levels H, I, and J are of 64 dimensions. For possible enhancement, we may explore the use of histograms of mixed dimensions. For example, in the Horizontal-and-Vertical Search, we can study the possible improvement (or degradation) using 512-dimensional color histograms for horizontal filtering on the first level and 8-dimensional color histograms for vertical filtering on the remaining levels.

Moreover, in the experiments for determining the appropriate number of levels in the multiscale representation and for finding the best search strategy, color histograms of 8, 64, and 512 dimensions are used. These color histograms were created for real images collected from various sources and covering wide application domains. However, the possibility of bias in particular domains may exist. Correspondingly, we can generate realistic random color histograms [Stricker 1994], and use the resulting histograms to strengthen our findings.

Bibliography

[Ashley *et al* 1995] Jonathan Ashley, Ron Barber, Myron Flickner, James Hafner, Denis Lee, Wayne Niblack, and Dragutin Petkovic. *Automatic and Semi-automatic Methods for Image Annotation and Retrieval in QBIC.* Research Report RJ 9951, IBM Almaden Research Center, San Jose CA, April 1995.

[Bach *et al* 1996] Jeffrey R Bach, Charles Fuller, Amarnath Gupta, Arun Hampapur, Bradley Horowitz, Rich Humphrey, Ramesh Jain, and Chiao-fe Shu. "The Virage Image Search Engine: An Open Framework for Image Management". *Proceedings of SPIE Conference on Storage and Retrieval for Still Image and Video Databases IV (Vol 2670)*: 76-87. San Jose CA, January-February 1996.

[Barber *et al* 1994] R Barber, M Flickner, J Hafner, D Lee, W Niblack, D Petkovic, J Ashley, T McConnell, J Ho, J Jang, D Berkowitz, P Yanker, M Vo, D Haas, D Lassig, S Tate, A Chang, P van Houten, J Chang, T Petersen, D Lutrell, M Snedden, P Faust, C Matteucci, M Raynor, R Peters, W Beck, and J Witsett. "Ultimedia Manager: Query By Image Content and its Applications". *Digest of Papers of the Spring COMPCON '94*: 424-429. San Francisco CA, February-March 1994.

[Beckmann *et al* 1990] N Beckmann, H-P Kriegel, R Schneider, and B Seeger. "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles". *Proceedings of ACM SIGMOD Conference on Management of Data*: 322-331. Atlantic City NJ, May 1990.

[Berchtold *et al* 1996] S Berchtold, D A Keim, and H-P Kriegel. "The X-tree: An Index Structure for High-dimensional Data". *Proceedings of the 22th VLDB Conference.* Bombay, India, September 1996.

[Cárdenas 1975] Alfonso F Cárdenas. "Analysis and Performance of Inverted Data Base Structures". *Communications of the ACM 18(5)*: 253-263. May 1975.

[Castelli *et al* 1997] Vittorio Castelli, Chung-Sheng Li, and Lawrence D Bergman. *Searching Image Databases at Multiple Levels of Abstraction.* Research Report RC 20702, IBM T J Watson Research Center, Yorktown Heights NY, January 1997.

[Chen *et al* 1997] Jau-Yuen Chen, Charles A Bouman, and Jan P Allebach. "Multiscale Branch and Bound Image Database Search". *Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases V (Vol 3022)*: 133-144. San Jose CA, February 1997.

[Chiueh 1994] Tzi-cker Chiueh. "Content Based Image Indexing". *Proceedings of the 20th VLDB Conference*: 582-593. Santiago, Chile, September 1994.

[Dimai and Stricker 1996] Alexander Dimai and Markus Stricker. *Spectral Covariance and Fuzzy Regions for Image Indexing*. Technical Report BIWI-TR-173, Swiss Federal Institute of Technology, Zürich, Switzerland, April 1996.

[Faloutsos *et al* 1994] C Faloutsos, W Equitz, M Flickner, W Niblack, D Petkovic, and R barber. "Efficient and Effective Querying by Image Content". *Journal of Intelligent Information Systems 3(3-4)*: 231-262. 1994.

[Faulus 1996] Dwifiandika S Faulus. *Design and Implementation of an Expressive Query Interface/Language for Image Database*. MSc Thesis, The University of British Columbia, August 1996.

[Faulus and Ng 1996] Dwifiandika S Faulus and Raymond T Ng. "EXQUISI: An Expressive Query Interface for Similar Images". *Proceedings of SPIE Conference on Storage and Retrieval for Still Image and Video Databases IV (Vol 2670)*: 215-226. San Jose CA, January-February 1996.

[Faulus and Ng 1997] Dwifiandika S Faulus and Raymond T Ng. "An Expressive Language and Interface for Image Querying". *Machine Vision and Applications 10(2)*: 74-85. 1997.

[Finn 1996] Robert Finn. "Querying By Image Content". *IBM Research 3*: 22-25. 1996.

[Flickner *et al* 1995] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. "Query by Image and Video Content: The QBIC System". *IEEE Computer 28(9)*: 23-31. September 1995.

[Gong *et al* 1994] Yihong Gong, Hongjiang Zhang, and Chua Hock Chuan. "An Image Database System with Fast Image Indexing Capability based on Color Histograms". *Proceedings of 1994 IEEE Region 10 Conference on Frontiers of Computer Technology*: 407-411. Singapore, August 1994.

[Gudivada and Raghavan 1995] Venkat N Gudivada and Vijay V Raghavan. "Content-based Image Retrieval Systems". *IEEE Computer 28(9)*: 18-22. September 1995.

[Gupta 1997] Amarnath Gupta. *Visual Information Retrieval Technology: A Virage Perspective.* Virage Inc, San Mateo CA, February 1997.

[Guttman 1984] Antonin Guttman. "R-trees: A Dynamic Index Structure for Spatial Searching". *Proceedings of ACM SIGMOD Conference on Management of Data*: 47-57. Boston MA, June 1984.

[Hirata *et al* 1993] Kyoji Hirata, Yoshinori Hara, Naoki Shibata, and Fusako Hirabayashi. "Media-based Navigation for Hypermedia Systems". *Proceedings of ACM Conference on Hypertext*: 159-173. Seattle WA, November 1993.

[Hurvich 1981] Leo Maurice Hurvich. *Color Vision.* Sinauer Associates, 1981.

[Ioka 1989] Mikihiro Ioka. *A Method of Defining the Similarity of Images on the Basis of Color Information.* Research Report RT 0030, IBM Tokyo Research Laboratory, Tokyo, Japan, December 1989.

[Jacobs *et al* 1995] Charles E Jacobs, Adam Finkelstein, and David H Salesin. "Fast Multiresolution Image Querying". *Proceedings of ACM SIGGRAPH Conference on Computer Graphics & Interactive Techniques*: 277-286. Los Angeles CA, August 1995.

[Jain 1993] R Jain (editor). "NSF Workshop Report on Visual Information Management Systems". *SIGMOD Record 22(3)*: 57-75. September 1993.

[Kurniawati *et al* 1997] R Kurniawati, J S Jin, and J A Shepherd. "The SS$^+$-tree: An Improved Index Structure for Similarity Searches in a High-dimensional Feature Space". *Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases V (Vol 3022)*: 110-120. San Jose CA, February 1997.

[Li *et al* 1995] C Li, J Turek, and E Feig. "Progressive Template Matching for Content-based Retrieval in Earth Observing Satellite Image Database". *Proceedings of SPIE Conference on Digital Image Storage and Archiving Systems (Vol 2606).* Philadelphia PA, October 1995.

[Lin *et al* 1994] King-Ip Lin, H V Jagadish, and Christos Faloutsos. "The TV-tree — An Index Structure for High-dimensional Data". *VLDB Journal 3(4)*: 517-549. October 1994.

[Luenberger 1984] David G Luenberger. *Linear and Nonlinear Programming.* Addison-Wesley, 1984.

[Niblack *et al* 1993] Wayne Niblack, Ron Barber, Will Equitz, Myron Flickner, Eduardo Glasman, Dragutin Petkovic, Peter Yanker, Christos Faloutsos, and Gabriel Taubin. *The QBIC Project: Querying Images by Content using Color, Texture, and Shape.* Research Report RJ 9203, IBM Almaden Research Center, San Jose CA, February 1993.

[Ng and Sedighian 1996] Raymond T Ng and Andishe Sedighian. "Evaluating Multi-dimensional Indexing Structures for Images Transformed by Principal Component Analysis". *Proceedings of SPIE Conference on Storage and Retrieval for Still Image and Video Databases IV (Vol 2670)*: 50-61. San Jose CA, January-February 1996.

[Ng and Tam 1997] Raymond T Ng and Dominic Tam. "An Analysis of Multi-level Color Histograms". *Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases V (Vol 3022)*: 22-34. San Jose CA, February 1997.

[Pentland *et al* 1994] A Pentland, R W Picard, and S Sclaroff. "Photobook: Tools for Content-based Manipulation of Image Databases". *Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases II (Vol 2185)*: 34-47. San Jose CA, February 1994.

[Petkovic *et al* 1996] Dragutin Petkovic, Wayne Niblack, Myron Flickner, David Steele, Denis Lee, John Yin, James Hafner, Frank Tung, Harold Treat, Richard Dow, May Gee, Mimi Vo, Peter Vo, Bonnie Holt, Janet Hethorn, Kenneth Weiss, Peter Elliott, and Colin Bird. *Recent Applications of IBM's Query By Image Content (QBIC)*. Research Report RJ 10006, IBM Almaden Research Center, San Jose CA, January 1996.

[Pope and Lowe 1994] Arthur R Pope and David G Lowe. "Vista: A Software Environment for Computer Vision Research". *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*: 275-280. Seattle WA, June 1994.

[Sakamoto *et al* 1994] Hiroaki Sakamoto, Hideharu Suzuki, and Akira Uemori. "Flexible Montage Retrieval for Image Data". *Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases II (Vol 2185)*: 25-33. San Jose CA, February 1994.

[Salton and McGill 1983] Gerard Salton and Michael J McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, 1983.

[Samet 1990] Hanan Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[Sawhney and Hafner 1993] Harpreet S Sawhney and James L Hafner. *Efficient Color Histogram Indexing for Quadratic Form Distance Functions.* Research Report RJ 9572, IBM Almaden Research Center, San Jose CA, October 1993.

[Sawhney and Hafner 1994] Harpreet S Sawhney and James L Hafner. "Efficient Color Histogram Indexing". *Proceedings of IEEE International Conference on Image Processing.* 1994.

[Schrage 1986] Linus E Schrage. *Linear, Integer, and Quadratic Programming with LINDO (Third Edition).* Scientific Press, 1986.

[Sedighian 1995] Andishe S Sedighian. *A Comparative Analysis of Multi-dimensional Indexing Structures for 'Eigenimages'*. MSc Thesis, The University of British Columbia, December 1995.

[Sellis *et al* 1987] Timos Sellis, Nick Roussopoulos, and Christos Faloutsos. "The R^+-tree: A Dynamic Index for Multi-dimensional Objects". *Proceedings of the 13th VLDB Conference*: 507-518. Brighton, England, September 1987.

[Sigmon 1992] Kermit Sigmon. *MATLAB Primer (Second Edition)*. Department of Mathematics, University of Florida, 1992.

[Sistla *et al* 1994] A Prasad Sistla, Clement Yu, and R Haddad. "Reasoning about Spatial Relationships in Picture Retrieval Systems". *Proceedings of the 20th VLDB Conference*. Santiago, Chile, September 1994.

[Sistla *et al* 1995] A Prasad Sistla, Clement Yu, Chengwen Liu, and King Liu. "Similarity Based Retrieval of Pictures using Indices on Spatial Relationships". *Proceedings of the 21st VLDB Conference*: 619-629. Zürich, Switzerland, September 1995.

[Smith and Chang 1996] John R Smith and Shih-Fu Chang. "Tools and Techniques for Color Image Retrieval". *Proceedings of SPIE Conference on Storage and Retrieval for Still Image and Video Databases IV (Vol 2670)*: 426-437. San Jose CA, January-February 1996.

[Stone and Li 1995] Harold S Stone and Chung-Sheng Li. *Image Matching by means of Intensity and Texture Matching in the Fourier Domain*. Technical Report 95-6, NEC Research Institute, Princeton NJ, July 1995.

[Stricker 1994] Markus A Stricker. "Bounds for the Discrimination Power of Color Indexing Techniques". *Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases II (Vol 2185)*: 15-24. San Jose CA, February 1994.

[Stricker and Dimai 1996] Markus Stricker and Alexander Dimai. "Color Indexing with Weak Spatial Constraints". *Proceedings of SPIE Conference on Storage and Retrieval for Still Image and Video Databases IV (Vol 2670)*: 29-40. San Jose CA, January-February 1996.

[Stricker and Orengo 1995] Markus Stricker and Markus Orengo. "Similarity of Color Images". *Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases III (Vol 2420)*: 381-392. San Jose CA, February 1995.

[Swain and Ballard 1990] Michael J Swain and Dana H Ballard. "Indexing via Color Histograms". *Proceedings of IEEE Third International Conference on Computer Vision*: 390-393. Osaka, Japan, December 1990.

[Tam 1996] Dominic Pok Man Tam. *An Analysis of Multi-level Filtering for High-dimensional Image Data.* MSc Thesis, The University of British Columbia, April 1996.

[Tao and Dickinson 1996] B Tao and B Dickinson. "Classified Template Matching for Satellite Image Retrieval". *Proceedings of 30th Annual Conference on Information Sciences and Systems.* Princeton NJ, 1996.

[Tao *et al* 1997] Bo Tao, Kevin Robbins, and Bradley Dickinson. "Image Retrieval with Templates of Arbitrary Size". *Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases V (Vol 3022)*: 2-11. San Jose CA, February 1997.

[Thomasian *et al* 1997] Alex Thomasian, Vittorio Castelli, and Chung-Sheng Li. *RCSVD: Recursive Clustering with Singular Value Decomposition for Dimension Reduction in Content-based Retrieval of Large Image/Video Databases.* Research Report RC 20704, IBM T J Watson Research Center, Yorktown Heights NY, January 1997.

[White and Jain 1996] David A White and Ramesh Jain. "Similarity Indexing with the SS-tree". *Proceedings of IEEE International Conference on Data Engineering.* New Oreleans LA, February 1996.

[White and Jain 1997] David A White and Ramesh Jain. "ImageGREP: Fast Visual Pattern Matching in Image Databases". *Proceedings of SPIE Conference on Storage and Retrieval for Image and Video Databases V (Vol 3022)*: 96-107. San Jose CA, February 1997.

[Yao 1977] S B Yao. "Approximating Block Accesses in Database Organizations". *Communications of the ACM 20(4)*: 260-261. April 1977.