UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

SEARCHING IMAGES BY COLOR IN MULTIMEDIA DATABASE SYSTEMS

A Dissertation

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

degree of

Doctor of Philosophy

By

LEONARD BROWN

Norman, Oklahoma
2003

SEARCHING IMAGES BY COLOR IN
MULTIMEDIA DATABASE SYSTEMS

A Dissertation APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

[illegible]
Le Gruenwald, Committee Chair

[illegible]
Sudarshan K. Dhall

[illegible]
S. Lakshmivarahan

[illegible]
Samuel Lee

[illegible]
K. Thulasirahan

ACKNOWLEDGMENTS

I would not have been able to complete this work without the help and support of many people, and I would like to take this opportunity to thank them. First, I want to thank my advisor, Dr. Gruenwald. She is the person who is the most responsible for the successful completion of this work due to here infinite patience and guidance. I would also like to thank my committee of Drs. Dhall, Lee, Thulasirahan, and Lakshmivarahan as well as Dr. Speegle from Baylor University for their valuable time and their thoughtful recommendations.

I also want to thank the members of the computer science community here at OU. I thank the past and present members of the OUDB research group for their support and friendship. I will always cherish the friends that I have made here at OU over the years including Brian, Carlos, David, Diana, Gary, Hongping, Javed, Jianting, Jim, Khushru, Lau, Leslie, Shankar, Sirirut, Sylvain, and Zahid.

I would also like to thank Wayne Steen and the Minority Engineering Program for supporting me throughout my undergraduate and graduate work. I would also like to thank Nicolas Pontier and Jeff Watkins for their contributions to the prototype.

Finally, I would also like to thank my family and friends who are more important to me than any degree. They supported me in my decision to return to school and always encouraged me not to give up my goals. I would not have completed this work without the support of my sister Kara, my brother-in-law James, my mother Rose, my aunts and uncles and their families, and my friends Michael and Ron. In addition, I want to thank my three-year-old niece Rachel for serving as the inspiration to complete my degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Previous research has demonstrated that instead of storing images in a Multimedia DataBase Management System (MMDBMS) using a conventional binary format, space can be saved by storing some of the images virtually, meaning that they are stored as sequences of editing operations. Since the existing techniques for searching images by color typically assume that the images are stored in conventional binary formats, new techniques and strategies for processing the queries are needed when the images are stored virtually. The goal of this dissertation is to develop techniques for performing color-based searches of virtual images and determine their strengths and weaknesses.

This dissertation presents several tasks that have been completed in order to achieve the above goal. First, this dissertation presents algorithms for processing color-based queries based on the colors contained within an image. They process queries of the type "*Identify all images that are between* PCT_{min} *and* PCT_{max} *percent of color* C_Q", where PCT_{min} and PCT_{max} represent percentages and C_Q represents a color in the RGB (Red, Green, Blue) model.

Next, this dissertation proposes algorithms for measuring the similarity between two images when one of them is stored virtually, where the similarity is based on the colors contained within an image. This allows an MMDBMS to process color-based searching queries of the type "*Identify the k images that most resemble Q based on color*", where k represents the desired number of images, and Q represents a query object by providing a method to measure how similar each virtual image is to the query object.

Third, this dissertation proposes a data structure for organizing virtual images identifiers stored in the MMDBMS in order to reduce the amount of time it takes to process the above algorithm. By using the data structure, the system will be able to identify some of the virtual images that can satisfy a given query without analyzing their sequences of editing operations. The reduction in the query processing time occurs from the reduction in the number of virtual images that have to be analyzed.

Finally, this dissertation constructs a prototype system to compare the above algorithms to the conventional approach for processing color-based search queries that use images stored as binary objects. The performance evaluation is based on permanent storage space used, color-based search query processing time, insertion query processing time, as well as accuracy. The comparison results show that unlike the alternative approaches, the proposed algorithms are able to perform efficiently in both searching and insertion time while still saving storage space through the use of virtual images.

CHAPTER 1

INTRODUCTION

1.1. Motivation

One of the most valuable assets today is information. Businesses and other organizations can gain competitive advantages from simply having the ability to store and retrieve large quantities of information quickly and efficiently. Consequently, there has been a tremendous amount of research devoted to the development of DataBase Management Systems (DBMSs) that perform these functions [Kort, 1991]. In recent years, however, organizations have had an increasing need to track images and other types of multimedia data.

Unfortunately, conventional DBMSs are not appropriate for storing and retrieving images ([Blan 1997], [Gros, 1997]). The reason is that images have different characteristics than the traditional alphanumeric data stored in conventional systems. The work in this dissertation focuses on the differences that arise in the area of storing images and retrieving them, which will be addressed in the following sections.

1.1.1. Storing Images

One characteristic that makes storing images more difficult than storing traditional alphanumeric data is that images require more space. For example, a single image object often requires several hundred kilobytes, while a very high resolution image can use several megabytes [Klas, 1997]. Even with the falling cost of memory, attempting to store thousands of these data objects can quickly exhaust a database's storage space.

The necessity of a method that reduces the space used by storing such images becomes evident by discussing some example applications. First, consider an application that stores images obtained from orbiting telescopes or space stations. When the raw images are received, they may be processed to create new versions that enhance certain points of interest. For example, in order to provide a more detailed view of a storm displayed in one of the images, a new image may be created by cropping the storm, and then enlarging it. The user, however, will want to save both the original image and the enlarged picture of the storm. Another application is one that allows an interior designer to decorate a room by editing a picture of it. The designer may change the color of the walls or carpet, add or remove furniture, and experiment with different lighting effects.

The above applications are examples where several similar images may be stored in the database, which means that these images would contain a lot of redundant data. As in traditional databases, such redundant data should be eliminated. One method for avoiding the storage problem described earlier is to eliminate the redundancy in these types of applications by changing how the data objects are stored ([Grue, 1996], [Spee, 1995], [Spee, 1998]). The idea is that instead of storing two similar images in their binary, space-intensive formats, only one, called the *base* image, is stored in that manner. The other, called a *derived* image, is stored as a reference to the first (base) image along with a set of editing operations used to transform it into the second one, as displayed in Figure 1-1. The derived image is therefore represented as a transformation of the base image. Displaying an image stored in this format can be accomplished by accessing the base image and sequentially performing the associated editing operations on it, which is a process called *instantiation* [Grue, 1996]. An image stored in this format is called a *virtual image.*

```
<Base Image>

<Operation 1> <Parameters>
<Operation 2> <Parameters>
            ...
<Operation n> <Parameters>
```

Figure 1-1. Description of a Virtual Image

As examples of this concept, consider Figures 1-2a and 1-2b. Both figures illustrate the effect of storing a pair of similar images in a system that uses virtual images. In Figure 1-2a, the base image is an image of the French flag, and the derived image depicts the Italian flag that was created by changing the blue pixels in the French flag to green. In a system that uses virtual images, the French flag would be stored normally, while the Italian Flag would be stored virtually, which includes a reference to the French Flag and the sequence of operations "select entire image" and "change blue pixels to green" that can be later used to instantiate the close-up. Alternatively, Figure 1-2b illustrates the difference in storing the storm photos described in the example application presented earlier. Again, in a system that uses virtual images, the base image would be stored normally, while the enlarged photo would be stored as a reference to the base image and the sequence of operations "select center", "crop selected region", and "enlarge selected region".

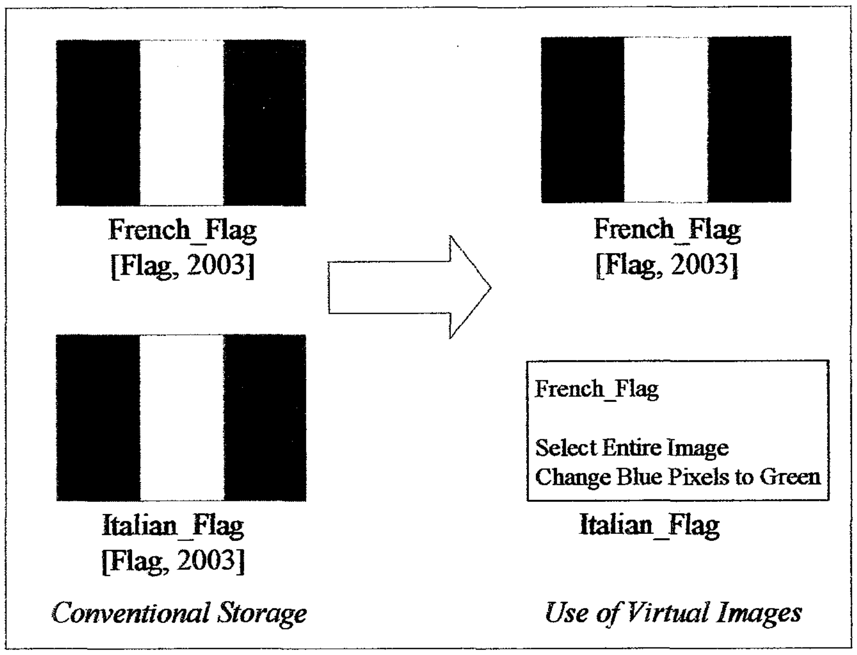


Figure 1-2a. Storage of Similar Flags with and without Virtual Images

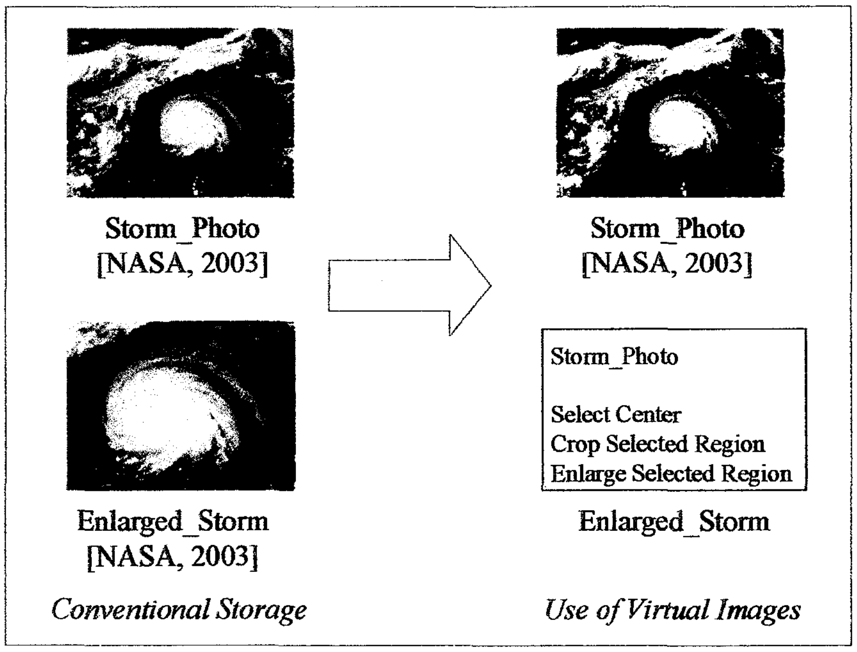


Figure 1-2b. Storage of Similar Photos with and without Virtual Images

In addition to using less space, using virtual images offers other advantages. Unlike many compression methods such as JPEG [Wall, 1991], storing and instantiating a virtual image is a lossless process. The derived object, then, can be retrieved endlessly without any degradation. Another advantage of virtual images is that they are not dependent upon any particular compression or storage format, nor are they dependent upon any particular computing platform. Thus, virtual images are portable.

1.1.2. Searching Images

Although virtual images address the storage requirements of MMDBMSs, other issues arise when considering image retrieval. One such requirement is that an MMDBMS should facilitate searching images based on their content, but it is insufficient, however, to represent that content using only textual descriptions such as filenames or keywords. To illustrate, consider a query requesting all images in the database that contain a picture of a dark blue sports car. A conventional DBMS would require that the keywords "dark", "blue", "sports", and "car" be attached to any image in order to return it as a result of this query. Not only does this require that humans inspect each image and attach keywords, but also it prohibits humans from using other phrases such as "navy automobile" or to make common judgment errors such as "black corvette".

Instead of using keywords that are manually associated with an image, an image ideally should be retrieved using automatically extracted features or attributes that represent its content. Features that are commonly extracted from images include color, texture, and shape. Systems that retrieve images using automatically extracted features are called *Content-Based Image Retrieval (CBIR)* systems [Eaki, 1998]. Examples of these systems include BIC [Steh, 2002], DISIMA ([Oria, 2001], [Oria, 2000]), MARS [Orte, 1998], QBIC ([Falo, 1994, Flic, 1995, Hafn, 1995]), ARTISAN ([Eaki, 1998], [Eaki, 1996]), FIBSSR [Mehr, 1995], and ImageRoadMap [Park, 1997].

CBIR systems typically use the following approach to retrieve images. As each image is entered into the database, the values of the features that can be used for querying are automatically identified. Each image, then, is represented by the set of values of the features extracted from it, called a feature vector. The result is that to search the set of images in the database in response to a query, the CBIR system can search all of the extracted feature vectors. To illustrate, consider a CBIR system that allows searching based on the colors in images. In such a system, a histogram can be created for each image where each bin contains the number of pixels of a particular color in that image. When normalized, each bin represents the percentage of pixels of a particular color in the image. So, as long as each image is represented by such a histogram, the users can query the database requesting the images that have a specified percentage of pixels containing a certain color. An example of such a query is "*Retrieve all images that are 25% blue.*" Similar histogram methods are used by numerous CBIR systems including ([Steh, 2000], [Djer, 1997], [Gray, 1995], [Hafn, 1995], [Orte, 1998], [Park, 1999], [Scla, 1997]).

Some queries that are commonly presented to a CBIR system request the images in the database that are most similar to an input query image. Queries of this type are called *nearest neighbor queries* ([Bozk, 1999], [Falo, 1996]). To process them, a feature vector is generated from the input query image, and is then compared to the feature vectors representing the stored images in the database. The similarity between the query image and an image in the database is determined by measuring the similarity between their two representative feature vectors. So, in response to a nearest neighbor query, the CBIR system returns the images corresponding to the feature vectors that are the most similar to the feature vector of the input query image.

Once the images and feature vectors are in the database, locating the nearest neighbors of a goal or query object has the worst case requirement of computing the distance between it and every other object in the database [Fagi, 1998]. Multidimensional indexes, such as the R-tree [Gutt, 1984] and its variants ([Brow, 1998a], [Gaed, 1998]) have been used to reduce the number of distance computations.

Because color is used so frequently in CBIR systems, this dissertation focuses on searching images using color. It should be noted, however, that there are other properties that are also used frequently. Many systems allow users to query images using feature vectors based on texture ([Djer, 1997], [Kell, 1995], [Smit, 1995]) and shape ([Boue, 1999], [Djer, 1997], [Eaki, 1998], [Park, 1997]). As with color, these feature vectors are extracted from the images in the database and are subsequently used to process nearest neighbor queries.

1.1.3. Problem Statement

The preceding sections indicate that two requirements of an MMDBMS are to store images efficiently and facilitate searching images based on their content, specifically color. A problem arises when using virtual images when considering searching because the existing techniques for searching images by color are based on images stored in a binary format. *The goal of this research is to develop algorithms for performing color-based searching of virtual images.* Specifically, this dissertation will propose algorithms for processing color-based range queries and performing similarity searches of virtual images using color histograms to represent image content.

This research in this dissertation is most suitable for applications in which users frequently create new images that are similar to each other, and all of the images in the database are searched by color only. An example of this type of application is an online retail-clothing database, which permits users to search for images of clothes that match colors contained in their real-world apparel. An example query in this application would be to "*Search for pants that match the colors contained in this tie*". This application would benefit from storing photos of similar clothing designs virtually. The similarity exists due to the fact that designers usually create multiple versions of the same designs.

Considering only color for searching images, the applicability of this research is limited since most of the real-world applications require retrieving of images based on other properties such as texture and shape in addition to color. However, since there is no existing work that addresses content-based search for images that are stored virtually, this research serves as a starting point for this area.

1.2. Organization of the Dissertation

Chapter 2 discusses the current research in performing CBIR in conventional multimedia database management systems.

Chapters 3-5 present the proposed techniques for performing color-based searching. Chapter 3 presents an algorithm for identifying the colors within a virtual image, Chapter 4 presents the algorithm for processing nearest neighbor queries, and Chapter 5 proposes a data structure that can be used for speeding up the query processing.

Chapter 6 proposes two additional approaches for searching images using color. Both approaches convert the virtual images into a binary format so that they can be searched using conventional techniques.

Chapter 7 presents an analysis of each proposed algorithm and compares it to the conventional approach for processing retrieval queries that use conventional approaches. The comparisons are based on expected permanent storage space, average insertion time, average searching time, and retrieval accuracy. In addition, Chapter 8 presents a prototype system used to verify the results from the performance evaluations. Finally, Chapter 9 concludes the dissertation and provides areas for future work.

CHAPTER 2

SURVEY OF RELATED RESEARCH

This dissertation focuses on performing image retrieval in a Multimedia DataBase Management System (MMDBMS) that uses virtual images. Consequently, it is related to two main areas of research. The first area concerns the existing techniques used to perform Content-Based Image Retrieval (CBIR) using color histograms. The second area contains the research that is related to the usage of editing operations in images. Both areas are reviewed in this chapter.

2.1. Content-Based Image Retrieval Systems

There are numerous CBIR systems that exist in the literature, but the retrieval techniques used by those systems are different than the query processing algorithms proposed in this research. The reason is that the proposed algorithms use the editing operations contained inside virtual images to improve retrieval efficiency. Still, there are some aspects of this research that are shared by those systems that retrieve only images stored in a conventional binary format. These aspects are reviewed in this section.

2.1.1. Identifying Features

The first aspect addressed in CBIR systems is identifying the features used to determine the content of an image. Three features that are used frequently are color ([Steh, 2002], [Anna, 2000], [Djer, 1997], [Hafn, 1995], [Orte, 1998], [Lin, 2001], [Scla, 1997]), texture ([Anna, 2000], [Djer, 1997], [Kell, 1995], [Mehr, 1995]), and shape

([Boue, 1999], [Djer, 1997], [Eaki, 1998], [Oria, 2000], [Park, 1997]) because they can be extracted from most images ([Asla, 1999], Park, 1997]).

The features that should be extracted from images depend on the expected user queries of the applications. For example, in the application called ARTISAN (Automatic Retrieval of Trademark Images by Shape ANalysis) [Eaki, 1998], users are expected to pose queries that search for images of trademarks in the database that are similar to an arbitrary input shape. Consequently, the features that the underlying MMDBMS must extract are based on the shapes contained in its data objects. In addition, since the trademarks stored in the database are typically black and white, the users are not expected to pose queries that look for trademarks containing certain colors. Thus, it is not important for ARTISAN's underlying MMDBMS to extract features based on color out of its trademarks. Similar issues arise in CBIR systems that retrieve features that are more specific to a particular set of images. For example, in applications used to compare pictures of people's faces ([Bach, 1993], [Wu, 1994]), users will want to retrieve faces from the database based on features such as eye color or nose length. Consequently, these features are used to determine the content of each image instead of color, shape, and texture.

This research developed query processing algorithms that use color for representing the content in an image. As a result, the proposed algorithms are for applications whose underlying databases contain images that are expected to be retrieved using color.

2.1.2. Feature Extraction Techniques

The next aspect of performing CBIR consists of the techniques used to extract the features from images. When retrieving images using shape-based features, CBIR systems typically extract and identify the boundary of each shape using some representation such as a Freeman or chain code [Gonz, 1993]. These codes use a numeric value to represent each direction encountered as it traces the boundary of a shape. Alternatively, to retrieve images using texture-based features, a CBIR system may compute one or more co-occurrence matrices for each image [Gonz, 1993], which count the number of times pixels with different intensities are positioned near each other.

The focus of this dissertation is on color-based features. For such features, one common method of searching images is to create a histogram for each image where each bin contains the number of pixels of a particular color in its corresponding image. For example, if a system stored only black and white images, it would need histograms with two bins, one representing black and the other representing white. Given an image with 10 total pixels, 7 of which were black, its corresponding histogram would have a value of 7 in the bin representing the black color and a value of 3 in the bin representing the white color.

Often images in a system are of different sizes, where the size of an image is its total number of pixels. In order to compare images, systems usually normalize their histograms, meaning that the total number of pixels in each bin is divided by the total number of pixels in the entire image. Each bin, then, represents the percentage of pixels in the image that contain its corresponding representative color. So, for the example black and white image used earlier, a normalized histogram would have a value of .7 in the bin representing the black color and .3 in the other bin. Thus, a histogram can be computed counting the number of pixels that occur for each color defined by a system, then normalizing these values by dividing each of them by the total number of pixels in the image ([Swai, 1991], [Smit, 1995]).

When each image is represented by a histogram, the users can query the database requesting the images that have a specified percentage of pixels containing a certain color, such as "*Retrieve all images that are at least 25% blue*". Similar histogram methods are used by numerous CBIR systems including ([Djer, 1997, [Gray, 1995], [Hafn, 1995], [Orte, 1998], [Park, 1999], [Scla, 1997]). An example of using the histogram method to retrieve images is displayed in Figure 2-1. In the figure, there are several images of flags stored along with example histograms extracted from them. In response to the query "*Retrieve all images that are at least 25% blue*", the system can directly access the percentage of pixels in each flag that is blue. As a result, the system would return the first and third flags.

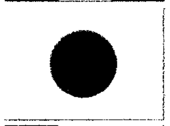
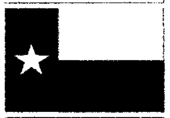
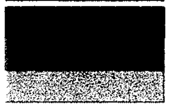
	Black	Blue	Yellow	Red	White
	0.0	0.33	0.0	0.33	0.33
	0.0	0.0	0.0	0.25	0.75
	0.0	0.30	0.0	0.33	0.34
	0.33	0.0	0.33	0.33	0.0

Figure 2-1. Example Histograms Extracted from a Set of Images [Flag, 2003]

Note from the above example that the number of bins in a histogram is directly related to the number of different colors recognized by the system. How the colors are identified varies from system to system. The different categories of variations [Smit, 1995] of identifying the colors used in histogram bins are reviewed in the remainder of this section.

2.1.2.1. Color Models

The first type of variation concerns the model used to represent the colors in an image. There are several models that express each color as a set of three or four values. Each of these expressed colors should correspond to one histogram bin. One of the most common models is the RGB (Red, Green, Blue) model, which represents each color as a combination of red, green, and blue wavelengths of light [Gree, 1995]. For example, combining red light and green light produces the color yellow, combining all wavelengths together produces the color white, and the absence of all wavelengths produces the color black. Typically, each wavelength can have a value between 0 and 255 [Gree, 1995], so the RGB color model can express a total of $(2^8)^3$ different colors.

Another common model is the CMYK model, which is an acronym for Cyan, Magenta, Yellow, and blacK [Gree, 1995]. Each color in this model is expressed as a combination of these four colors. The results of these combinations mimic the results that occur when combining ink [Gree, 1995]. Thus, the color black is obtained by adding high values of each of the four colors of the model, while white is represented as zero values for each of them.

Other color models were designed to separate the values that specify the tint of a color from the values that specify how light or dark it is [Gonz, 1993]. One example of such a model is the Hue, Saturation, Value (HSV) model ([Park, 1999], [Orte, 1998]). The first value, Hue, is represented as a value from 0 to 359, which indicates the tint of the color. For example, the color red is represented as a value of 0, and the color blue is represented as 240. The Saturation axis represents the amount of gray in the color. The last component, Value, represents how light or dark the color is. Another such model is the Luv color model developed by the Commission Internationale de l'Eclairage (CIE) [Park, 1999]. The first value, L, represents how light or dark the color is. The remaining two values, u and v, are combined to represent the tint of the color.

2.1.2.2. Quantization

The second variation to identifying the colors used in the CBIR system is the method used to quantize the color space represented by the selected model. As stated earlier, the RGB model expresses a total of $(2^8)^3$ different colors. The result is that a histogram that tracks the number of pixels that contain each of these expressed colors will have 16M bins. To reduce the number of bins in the histogram, several colors can be grouped together. For example, the colors with the RGB values of (0, 0, 0) and (0, 0, 1) can be grouped together in the same bin since humans cannot perceive the difference between them. An important issue, then, is how to quantize the color space in order to group similar colors together since many distinct values in the various color models are perceptually similar.

$$\left(\left\lfloor \frac{I_1}{\left(\frac{CM_1}{d_1}\right)} \right\rfloor \times d_2 \times d_3\right) + \left(\left\lfloor \frac{I_2}{\left(\frac{CM_2}{d_2}\right)} \right\rfloor \times d_3\right) + \left(\left\lfloor \frac{I_3}{\left(\frac{CM_3}{d_3}\right)} \right\rfloor\right)$$

Figure 2-2. Quantization Function for Mapping Color Model Values to Histogram Bins

A common method of quantizing the color space is to evenly divide each axis using some system-defined number of partitions, called uniform quantization [Park, 1999]. A formula for quantizing values of a 3-axis color model such as RGB or HSV is displayed in Figure 2-2. The method used in the formula is to divide each axis in the model into equal-sized partitions, and then assign a group number for an intensity value based on the set of partitions that contain it. In the formula, each d_i represents the number of partitions used for the ith axis, so the number of partitions in the first axis is d_1, the number of partitions for the second axis is d_2, and the number of partitions in the third axis is d_3. Note that the total number of different combinations of partitions are $D = d_1 \times d_2 \times d_3$. Since each combination of partitions maps to a single histogram bin, there will be D number of bins in the resulting histogram.

Let CM_i represent the number of different values that can be used for the ith axis of the color model of the system. For example, in the HSV model, the first axis, H, can have a value from 0 to 359. So, it can have 360 different colors. Each of the last two axes, S and V, can have values from 0 to 100, which means they can each have 101 different colors. So, if the system uses the HSV model, CM_1 equals 360, CM_2 equals 101, and CM_3 equals 101.

Given the above variables, the formula displayed in Figure 2-2 maps an intensity value in a 3-axis model to a single bin number. Let the intensity value be (I_1, I_2, I_3) where each I_i represents the value for the i^{th} axis in the model. To illustrate the use of the formula, let the system use the HSV color model where, as indicated before, (CM_1, CM_2, CM_3) = (360, 101, 101). Now, assume the system wants to divide the color space of the color model into 15 divisions along the H axis, 9 divisions along the S axis, and 9 divisions along the V axis as in [Park, 1999]. This means that (d_1, d_2, d_3) = (15, 9, 9). With these values, applying an HSV color (I_1, I_2, I_3) = (240, 0, 50) to the formula yields a value of floor(240/360/15)×81 + floor(0/101/9)×9 + floor(50/101/9) = 819.

System, Reference	Color Space	Color Space Partitioning
BIC ([Steh, 2002], [Steh, 2000])	RGB	R – 4 G – 4 B – 4
MARS ([Orte, 1998], [Orte, 1997])	HSV	H – 8 S – 4 V – 0
VisualSEEK, ([Smit, 1996], [Smit, 1996])	HSV	H – 18 S – 3 V – 3
[Pass, 1996]	RGB	R – 4 G – 4 B – 4
[Gray, 1995]	CIE-Luv	L – 8 u – 8 v – 8
QBIC, ([Hafn, 1995], [Flic, 1995])	RGB, HVC	R – 16 G – 16 B – 16
[Gong, 1994]	HVC	H – 8 V – 8 C – 8

Table 2-1. Partitioning Methods of CBIR Systems that Use Color Histograms

Table 2-1 provides a list of partitioning methods utilized by systems that use color histograms to retrieve images. The first column contains the names of the image retrieval systems and their associated references. The second column describes the reported color space used to retrieve images. The third column describes the reported number of times each axis in the color space was divided during testing or implementation. An entry R – 4, G – 8, B – 6 means that the Red axis in the RGB model was divided into 4 sections, the Green axis was divided into 8 sections, and the Blue axis was divided into 6 sections, which would create a histogram with (4 × 8 × 6) = 192 bins.

2.1.3. Feature Representation

Once the desired features have been extracted from each image, the next aspect to address is how to represent the features in the CBIR system. Many systems compute a set of descriptors, which are properties of the features extracted from each image. Determining the similarity of features, then, can be performed by comparing their corresponding sets of descriptors. To illustrate, consider the QBIC (Query By Image Content) system developed by IBM ([Falo, 1994], [Flic, 1995], [Hafn, 1995]). After the system identifies the shape of an object in an image, QBIC computes an 18-dimensional set of descriptors of the shape that includes such properties as the number of pixels contained in the entire shape called the area, and the number of pixels contained in the shape's boundary called the perimeter. Alternatively, for texture, QBIC computes a 3-dimensional set of descriptors, which represent the coarseness, contrast, and directionality of the texture within an image [Falo, 1994]. Other descriptors computed from a co-occurrence matrix representing the texture of an image include the maximum value in the matrix, called the maximum probability, and the sums of the squares of the matrix values, called the uniformity [Gonz, 1993].

For color, some systems compute and store a representative set of values based on the color histograms extracted from an image. For example, [Gong, 1994] only stores the 20 histogram bins with the most pixels. In VisualSEEK ([Smit, 1995], [Smit, 1996]), instead of storing a histogram, the system stores the set of colors that appear most frequently in a given image, which is referred to as a Color Set. The system in [Pass, 1996] uses Color Coherence Vectors (CCVs), which are histograms where each bin contains two values instead of one. The first value represents the number of pixels in the bin that are a part of an object in the image, and the second value represents the number of pixels that are not.

2.1.4. Defining Image Similarity

The next aspect of performing CBIR is to define some criteria for satisfying a similarity search, which means that there must be some method of defining how similar one image is to another one. In systems that represent features using a vector of values, such as a color histogram, each image's vector represents a single point in some multidimensional space ([Cheu, 1998], [Falo, 1996], [Gros, 1997]). A common approach for those systems is to use a metric axiom based function [Sant, 1999] that computes the distances between two such multidimensional points as a basis to measure similarity. For two normalized n-dimensional vectors, **x** = (x_1, ..., x_n) and **y** = (y_1, ..., y_n), typical examples of metric functions include the L_p Distances, $\sqrt[p]{\sum_{i=1}^{n}(x_i - y_i)^p}$ [Jaga, 1997], and the Histogram Intersection, $\sum_{i=1}^{n}\min(x_i, y_i)$ [Swai, 1991]. Another category of similarity measurements called set-theoretic functions computes the similarity as a function of the numbers of features that are identical in both images, different in both images, and contained in one image but not in the other [Sant, 1999].

2.1.5. Access Methods

Another important, but often overlooked, aspect of performing content-based image retrieval is the access method used to speed-up query processing. When performing CBIR using color histograms, it should be possible to retrieve the images based on any of the bins in the histogram. Traditional indexes for relational DBMSs like the B-tree and its variants [Come, 1979] are insufficient because they use a single value to represent a data record. Consequently, multiple indices would have to be created and maintained by the system in order to search images using different histogram bins. It is also not desirable to create a single key to represent each histogram by concatenating the values in its bins. The reason is that it would be difficult to search for images based on the bins listed last in the concatenated keys [Kuma, 1994].

Due to the above problems in using a B-tree for multidimensional data, much of the existing research for indexing CBIR systems surrounds developing multidimensional access methods. One of the more popular categories of such data structures contains variations of the R-tree [Gutt, 1984] where each node corresponds to a section of a multidimensional data space. The idea is that multidimensional vectors that are similar to each other should be near one another in the data space. These indices divide the data space into several regions and provide algorithms to let the query processing module quickly identify the regions that contain vectors that satisfy a given retrieval query. The technique used to identify the regions is to correlate them to nodes in a tree where a node and its descendants correspond to a region and its subregions.

Many variations of the R-tree are listed in ([Brow, 1998a], [Falo, 1996], [Gaed, 1998]). These variants can be categorized based on how they partition the data space. One category divides the entire multidimensional space into a grid. Examples include the K-D-B-tree [Robi, 1981], hB-tree [Lome, 1990], G-tree [Kuma, 1994], BV-tree [Free, 1995], and MB$^+$-tree, [Dao, 1996]. These trees differ from each other by the methods they use to divide the data space along each of its dimensions. Another category clusters the data together using Minimum Bounding Regions [MBRs], which includes the R$^+$-tree [Sell, 1987], R*-tree [Beck, 1990], P-tree [Jaga, 1990], TV-tree [Lin, 1994], X-tree [Berc, 1996], SS-tree [Whit, 1996], and SR-tree [Kata, 1997]. This category of trees differ from the previous one in that by computing the MBR of the vectors stored in each node, its trees only partition the portion of the multidimensional space where data elements exist. This allows them to eliminate large portions of unused space quickly. Their common disadvantage is that MBRs must be computed and maintained for each of their internal nodes. The third category of multidimensional indexes differs from the first two in that its trees cluster the vectors based only on how similar they are to each other. This category includes the VP-tree [Yian, 1992], GNAT [Brin, 1995], M-tree [Ciac, 1997], and MVP-tree ([Bozk, 1999], [Bozk, 1997]).

A database management system that uses virtual images will contain images stored conventionally as well. Thus, the system must contain techniques for retrieving conventional images as well as virtual images, which means that it must address each of the above aspects.

2.2. Virtual Image Editing Operations

This dissertation proposes a method of using the semantic information contained within the editing operations of virtual images to enhance CBIR. One task necessary to implement this research is to define the set of editing operations that may be used in the virtual images. This research adopts the set of editing operations that have been suggested to handle virtual images in database management systems ([Grue, 1996], [Spee, 1995], [Spee, 1998]). The set of operations are based on an algebra [Ritt, 1996] for images similar in function to the relational algebra for conventional data. The set consists of five editing operations called Define, Modify, Combine, Mutate, and Merge. The description and implementation of these operations as according to [Hu, 1999] follow in the subsequent paragraphs.

2.2.1. Define (x_1, y_1, x_2, y_2)

The Define operation does not change an image by itself, however it is used to identify regions in an image that will then be edited by subsequent operations. For example, to edit a particular region of an image, the Define operation first identifies the region, and then the following operations perform the actual changes. In this research, the implementation of the Define operation restricts it so that only rectangular regions may be identified.

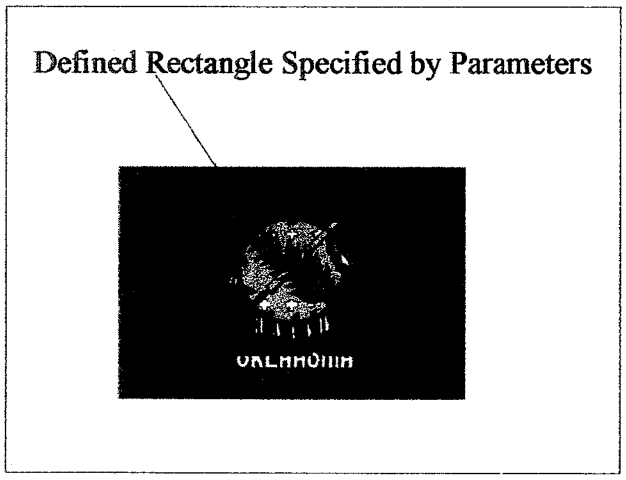


Figure 2-3. Rectangle Corresponding to Define(32, 96, 224, 288) [Flag, 2003]

The parameters of the Define operation identify the upper left (x_1, y_1) and lower right (x_2, y_2) coordinates of the rectangular region, which is called the Defined Rectangle

(DR). For example, Define (60, 100, 200, 250) specifies a rectangle ranging from the coordinates (60, 100) to (200, 250). An example of the rectangle created by the Define operation is displayed in Figure 2-3 where the dotted rectangle indicates the DR.

2.2.2. Mutate (M_{11}, M_{12}, M_{13}, M_{21}, M_{22}, M_{23}, M_{31}, M_{32}, M_{33})

The Mutate operation changes the positions of the pixels within an image. The 2-D coordinates (x, y) of each pixel are mapped into a 3-D vector $(x, y, 1)^t$. This vector is then multiplied by a 3x3 mutation matrix, which is specified in the parameter of the Mutate operation. This produces a new set of coordinates $(x', y', 1)^t$ for the pixel, which translate to the image position (x', y').

$$\begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} = \begin{bmatrix} 2.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Figure 2-4. Matrix Corresponding to Mutate (2, 0, 0, 0, 1, 0, 0, 0, 1)

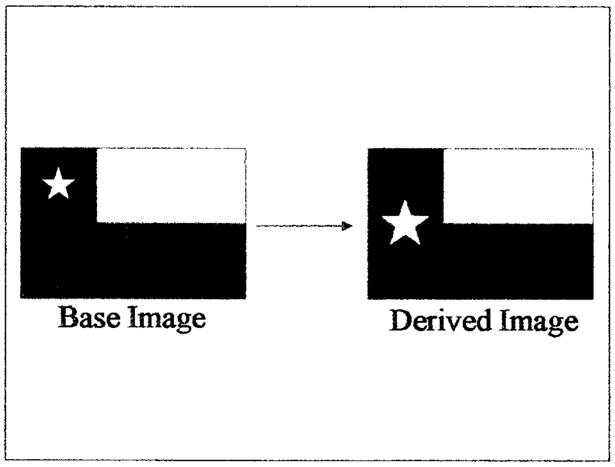


Figure 2-5. Effects after Applying Mutate(2, 0, 0, 0, 1, 0, 0, 0, 1) [Flag, 2003]

The Mutate operation can be used to perform combinations of rotations, scaling, and translation operations [Gonz, 1993]. To provide a specific example of the implementation of the Mutate operation, Mutate (2.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0) multiplies the $(x, y, 1)^t$ vector of the coordinates of each pixel in the current DR by the matrix displayed in Figure 2-4. This matrix scales the DR by a factor of 2 in the x-direction. Figure 2-5 illustrates the results of applying this operation on a DR enclosing the star in the upper left corner of the base image.

2.2.3. Modify ($\mathbf{Red_{min}}$, $\mathbf{Red_{max}}$, $\mathbf{Red_{new}}$, $\mathbf{Green_{min}}$, $\mathbf{Green_{max}}$, $\mathbf{Green_{new}}$, $\mathbf{Blue_{min}}$, $\mathbf{Blue_{max}}$, $\mathbf{Blue_{new}}$)

The Modify operation changes the colors of the pixels in a defined region as illustrated in Figure 2-6 which changes all blue pixels in the DR to green. It changes only those pixels in the defined region whose intensity values are within a specified range. Consequently, the parameters of the Modify operation describe the range of intensity values and the new color. Assuming that the intensity value of each pixel has a Red, Green, and Blue component, the parameters specify a new color and a range of values for each of the three components. So, there are 9 parameters to the Modify operation called R_{min}, R_{max}, R_{new}, G_{min}, G_{max}, G_{new}, B_{min}, B_{max}, and B_{new}, where the operation changes the color of only those pixels whose red intensity component is between R_{min} and R_{max}, green intensity component is between G_{min} and G_{max}, and blue intensity component is between B_{min} and B_{max}. The new color of the pixels after applying the operation will be (R_{new}, G_{new}, B_{new}). As an example in the RGB color model, Modify ((0, 50, 200), (75, 125, 100), (200, 255, 25)) changes all pixels in the current Defined Rectangle that have a Red axis component between 0 and 50, a Green axis component between 75 and 125, and a Blue axis component between 200 and 255 to (200, 100, 25).

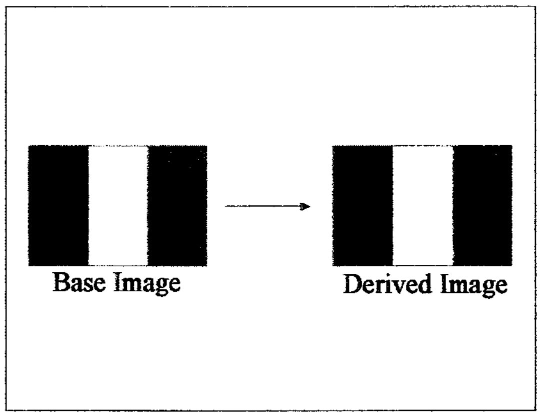


Figure 2-6. Effects after Applying Modify(0, 0, 0, 0, 0, 255, 255, 255, 0) [Flag, 2003]

In order to allow the tinting of an image, the implementation of the Modify operation allows one or more of the color axes to remain unchanged. For example, one method of tinting an image red is to change the Red intensity component to 255 and leaving the blue and green intensities unchanged. To specify that an intensity component should not be changed, the parameters set the minimum value of the range greater than the maximum value. As another example in the RGB color model, Modify ((0, 255, 255), (1, 0, 0), (1, 0, 0)) changes all pixels in the current DR that have a Red axis component between 0 and 255, which will be all pixels. After applying this example, a pixel with the color (R_x, G_x, B_x) will be changed to have the color (255, G_x, B_x).

2.2.4. Combine (C_{11}, C_{12}, C_{13}, C_{21}, C_{22}, C_{23}, C_{31}, C_{32}, C_{33})

The Combine operation can be used to blur images as displayed in Figure 2-7. Similarly, to the Modify operation, the Combine operation changes the intensity values of the pixel in the DR. Unlike Modify, however, it computes the pixel's new value to be the weighted average of its intensity and the intensities of its 8-neighbors. The weights are supplied as the parameters. Figure 2-8 shows an example of applying Combine (1, 2, 1, 2, 4, 2, 1, 2, 1) that computes the new intensity value of each pixel in the DR as the average of its neighbors using the matrix of weights displayed in Figure 2-7.

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Figure 2-7. Matrix of Weights Corresponding to Combine (1, 2, 1, 2, 4, 2, 1, 2, 1)

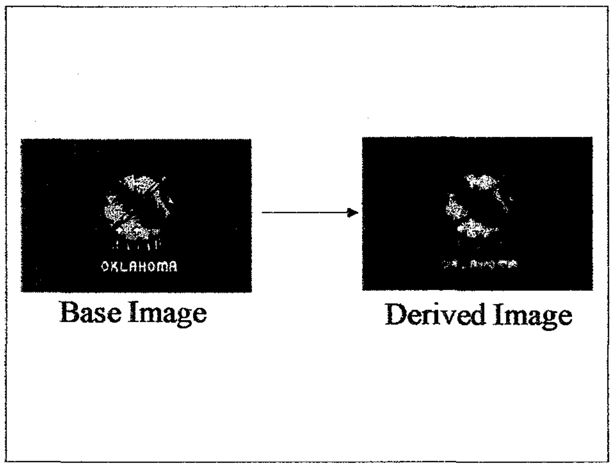


Figure 2-8. Effects after Applying Combine(1, 2, 1, 2, 4, 2, 1, 2, 1) [Flag, 2003]

2.2.5. Merge (Target Image, x_P, y_P)

Unlike the previous operations, Merge is a binary operation. It is used to combine two images as illustrated in Figure 2-9. The contents of the current Defined Rectangle are copied onto a target image starting at a given coordinate position (x_P, y_P). The coordinate position and the name of the target image are the parameters of the Merge operation. An example is Merge (image2, 100, 120), which copies the current DR onto image2 beginning at the point (100, 120). The crop operation can be implemented using this operation by specifying an empty target image. This has the effect of copying the current DR onto an empty image.

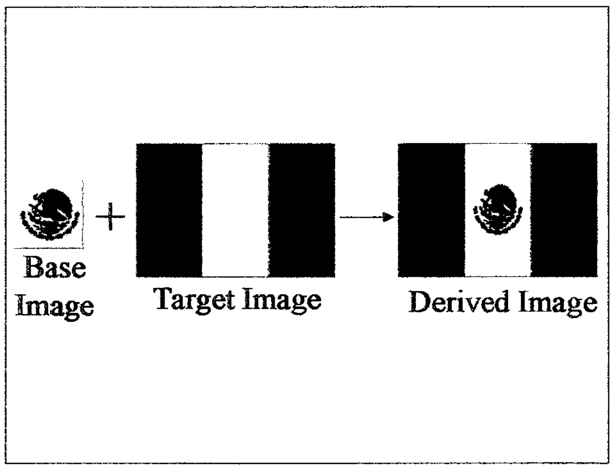


Figure 2-9. Effects after Applying Merge(image2, 100, 120) [Flag, 2003]

2.3. Virtual Image Retrieval

The work in [Aars, 1999] determines whether a virtual image satisfies a retrieval query by identifying the features of the image at the time that it is inserted into the database. Those features are then used to process the subsequent retrieval queries. In addition, the technique attaches a score to each feature representing the amount of it that it is contained in the image. This allows the technique to determine when features are present in an image even after another image is pasted on top of it as a result of the Merge operation. The score ranges from 0 to 100 where a higher score indicates that more of the feature is contained in the image.

CHAPTER 3

TECHNIQUES FOR PROCESSING COLOR-BASED RANGE QUERIES

This chapter proposes an algorithm that can be used to process color-based range queries in a MMDBMS that uses virtual images. The specific type of range query that the algorithm addresses is "*Retrieve all images that are between PCT_{min} and PCT_{max} percent of color C_Q*", where $\mathrm{PCT}_{\mathrm{min}}$ and $\mathrm{PCT}_{\mathrm{max}}$ represent percentages and C_{Q} represents a color in the RGB model. This type of query is useful for searching databases that contain images of objects that users want to access by color. Examples of these databases include clothing apparel, where users may pose the query "Retrieve all images of ties that contain no red" or "Retrieve all images of accessories that are dark brown", and automobiles where users may pose the query "Retrieve all images of cars that are at least 50% white".

As described in Chapter 2, one method of processing the query used in many existing systems ([Djer, 1997, [Gray, 1995], [Hafn, 1995], [Orte, 1998], [Park, 1999], [Scla, 1997]) is to extract the color histogram from each image and store it in the underlying database management system. The extracted histograms are then searched in response to a retrieval query, allowing the system to return the images that correspond to the selected histograms. Note that this means the system should have the ability to identify the image that corresponds to a given histogram, which can be accomplished by storing an image id in each histogram.

This approach uses the above color histogram technique to handle images stored conventionally, but proposes an alternative technique for retrieving virtual images. This is because extracting histograms from virtual images can be inefficient for two reasons. First, the existing feature extraction techniques typically operate on images stored in a binary format, so the virtual images must be instantiated before their features can be extracted. This can be inefficient since the instantiation process is slow. Second, extracting and storing the features of virtual objects along with the features of their bases may result in storing redundant data.

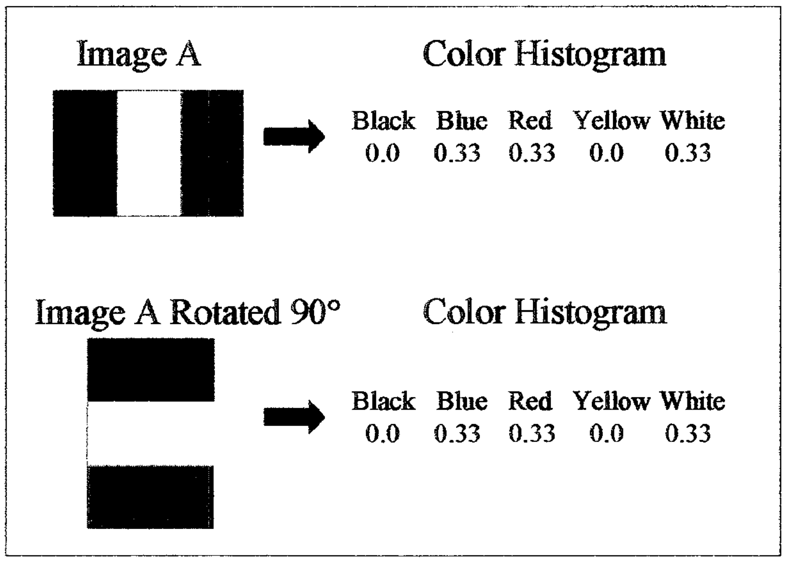


Figure 3-1. Sample Histograms Extracted from Similar Images [Flag, 2003]

To illustrate the second reason, consider Figure 3-1, which contains two similar images with the second image being created by rotating the first. Rotating an image only changes the locations of its pixels, meaning that it does not change the intensity values of the pixels. Consequently, both similar images should contain the same distribution of colors, which implies that the color histograms extracted from both should be the same. Since the histograms are the same, it would be redundant to store both of them. If the second image were stored virtually, then the rotation operation that was used to create it would be contained within its description. By being able to identify that an image is a rotated version of another image, the underlying database management system can infer during the processing of a retrieval query that the color histogram of the rotated image is identical to the histogram of the first image. This would allow the system to save space by storing only the color histogram of the first image.

The previous example illustrates the overall strategy used in this approach for processing range queries in virtual image retrieval systems. To be able to save storage space by using virtual images and at the same time avoid performance degradation due to image instantiation, this approach proposes to search images based on color using the semantic information in the description of virtual images instead of using the conventional approach of feature extraction. Since this semantic information is composed of a reference to a base image and a sequence of editing operations along with their parameters, the proposed searching algorithm is dependent upon a specific set of image editing operations. Because of this, it is important to identify desirable properties of such a set, which is performed in the next section. The remainder of the chapter is organized as follows: Section 3.2 describes the algorithm for processing range queries which requires the query processor to identify the colors of a virtual image, which is described in more detail in Section 3.3. Section 3.4 describes the rules used to develop the algorithm in Section 3.3, and Section 3.5 provides an example of the entire algorithm.

3.1. Properties of a Set of Image Editing Operations

The work in this research uses the set of five operations called Define, Modify, Mutate, Merge, and Combine ([Grue, 1996], [Spee, 1998], [Spee, 2000]) presented in Chapter 2. The following subsections describe algorithms for identifying and testing for desirable properties for such a set of image editing operations.

3.1.1. Ability to Transform Images

Since the sequence of editing operations used in a virtual image are for transforming a base image into another one, the one property of a set of image editing operations involves its ability to perform this transformation. Specifically, the set of operations should be able to transform any given image into another [Brow, 1997]. The proposed method to test for this property is based on reducing the definition of an image to a set of pixels where a pixel is defined as an entity with a 2-dimensional (x, y) location and an intensity value in the 3-dimensional RGB color model. Given this definition of an image, it is possible to convert one image, A, to another, B, by applying the image transformation algorithm displayed in Figure 3-2.

```
Step 1.  Let |A| represent the number of pixels in image A
Step 2.  Let |B| represent the number of pixels in image B
Step 3.  if |A| ≥ |B|
Step 4.          remove |A| – |B| pixels from image A
Step 5.  else
Step 6.          add |B| – |A| pixels to image A
Step 7.  For each pixel in A
Step 8.          Set location of pixel in A to match corresponding pixel in B
Step 9.          Set intensity of pixel in A to match corresponding pixel in B
```

Figure 3-2. Algorithm for transforming image A into image B

The transformation algorithm of changing image A into image B is decomposed in the above figure into steps that can be performed by common image editing operations, and it consists of two major tasks. The first task is to alter the number of pixels in image A so that it has the same number of pixels in image B, which is accomplished in steps 1 through 6. This allows each pixel in A to have a corresponding pixel in B. The second task is to modify each pixel in A so that it has the same intensity value and location of its corresponding pixel in B, which is accomplished in steps 7 through 9.

By demonstrating that a set of editing operations can be used to perform the image transformation algorithm, the set is shown to have the ability to transform a given image into any other image. There are three major steps that alter an image in the algorithm, which are adding a pixel, removing a pixel, and modifying a pixel. The ability to modify a pixel consists of two steps itself, changing its location and changing its intensity value. So, one method of demonstrating that the set of editing operations can perform any transformation from one image to another is to demonstrate that it can be used to perform all 4 major steps, adding an image, removing an image, changing a pixel's location, and changing a pixel's intensity.

Consider applying the above test to the set of editing operations Combine, Mutate, Merge, and Modify. Pixels can be removed using the Merge operation by merging a DR that consists of all of the pixels in the image except the ones to be removed onto an empty image. Pixels can also be removed using the Mutate operation by moving the pixels in the DR onto other pixels in the image. The same operations can be used to add pixels to an image as well. The Merge operation accomplishes this by pasting an image onto another image in a position that causes the second image to extend its borders.

Alternatively, the Mutate operation adds pixels by enlarging the DR. Changing the location of a pixel can be performed by the Mutate operation, which moves pixels from one (x, y) location to another. In addition, changing the intensity value of a pixel can be performed by either the Modify operation or the Combine operation as described in Chapter 2.

The above information, summarized in Table 3-1, implies that the set Combine, Mutate, Merge, and Modify can perform each of the four major tasks of the image transformation algorithm, which means that they can be used to transform an image into any other image. Note that the Define operation was not tested since it only specifies pixels within an image and does not actually alter it.

Image Transformation Algorithm Steps	*Corresponding Image Editing Operations*
Add a pixel to an image	Merge or Mutate
Remove a pixel from an image	Merge or Mutate
Change location of a pixel	Mutate
Change intensity of a pixel	Modify or Combine

Table 3-1. Operations Used to Perform Steps of Image Transformation Algorithm

3.1.2. Minimizing the Set of Editing Operations

Since, as demonstrated in the last section, the set of editing operations, Combine, Mutate, Merge, and Modify can be used to transform an image into any other image, the algorithms proposed in this research retrieve virtual images composed of only those operations. Another potentially desirable property of such a set is that all of the elements in the set are needed for it to perform all possible image transformations. A set that has this property is called *minimal*. Thus, for a set of image editing operations to be minimal, no subset of it can be used to perform all possible image transformations. As with the previous property, this research has developed a test to determine if a set of editing operations is minimal [Brow, 1998]. Here, a set of editing operations is defined to be minimal if the set can perform each of the steps in the image transformation algorithm and no proper subset of the set has this property.

Step 1:	Let S be a set $\{O_1, O_2, ..., O_N\}$, where each O_i is an image editing operation.
Step 2:	Show S can perform every step in the transformation algorithm
Step 3:	For i = 1 to N
Step 4:	Let S' = S – $\{O_i\}$.
Step 5:	Show S' cannot perform all steps in the image transformation algorithm

Figure 3-3. Algorithm for Testing if a Set of Editing Operations is Minimal

From the previous observation, a method of testing if a set of editing operations is minimal is to first demonstrate that it can perform all of the steps in the image transformation algorithm. The next task is to remove a single operation from the set and test to see if the remaining set of operations can perform all of the steps of the algorithm. If it can, then the original set is not minimal. This task must be repeated for each of the editing operations in the image set. This algorithm is presented in Figure 3-3.

Consider testing whether the set of editing operations, Combine, Mutate, Merge, and Modify is minimal. Since the set can perform each of the steps in the image transformation algorithm, the remaining task is to remove each operation individually and test the remaining set of operations against the transformation algorithm. When the Combine operation is removed from the set, the subset of the three remaining editing operations, Merge, Mutate, and Modify can perform each of the four tasks required by the image transformation algorithm as illustrated in Table 3-1. Thus, the original set of five editing operations is not minimal.

Alternatively, consider a set of operations consisting of only Mutate and Modify. To apply the testing algorithm of Figure 3-3 to the set, it is necessary to first demonstrate that the set can be used to perform all image transformations. From Table 3-1, the Mutate operation can add a pixel, remove a pixel, and change a pixel's location while the Modify operation can change a pixel's intensity value. So, Mutate and Modify can perform all image transformations.

The next step of the testing algorithm is to determine if any subset of Mutate and Modify can perform all image transformations. When Mutate is eliminated, only the Modify operation remains. That operation cannot add a pixel to an image, so it cannot perform all possible image transformations. When Modify is eliminated from the testing set, only the Mutate operation remains. This operation cannot change the intensity value of a pixel, which means that it also cannot perform all possible image transformations. Thus, no subset of Mutate and Modify can perform all image transformations, which means that the set is minimal.

When developing a query processing algorithm for virtual images, there are potential advantages for using a minimal set of editing operations such as simplifying the query processor and optimizer by reducing the number of possible operations. This research developed a query processing algorithm for the original set of operations Define, Combine, Mutate, Merge, and Modify, however. The reason is that the number of editing operations within each virtual image may become too large if the set of recognized operations is restricted.

To illustrate the above reason, consider blurring an image through the use of the Combine operation. Each pixel in the DR is changed to a new value that is computed using the intensities of its neighbors. Thus, several pixels with identical intensity values may be changed to different intensity values in a single Combine operation. Alternatively, the Modify operation can only change pixels with identical intensity values to the same new value, so it would take several different Modify operations to duplicate the blurring effect.

3.2. Algorithm for Processing Range Queries

The proposed algorithm is based on defining rules for determining how editing operations affect the colors contained in a virtual image if it is instantiated. Specifically, the algorithm determines how many pixels of the query color may be added to or removed from the base image after applying each of the associated editing operations to it. One of the benefits of this approach is that the proposed algorithm can be used to identify colors in a virtual image that are not present in its base image.

As a real-world example, consider Figure 3-4 in which the flag of Italy is stored as a transformation of the flag of France. The description of the Italian flag changes all of the blue pixels in the French flag to green. The result is that the image stored virtually will have green pixels in it, while its base image will only have red, white, and blue pixels. The proposed algorithm can identify that there are green pixels in the virtual image by examining the parameters of the Modify operation.

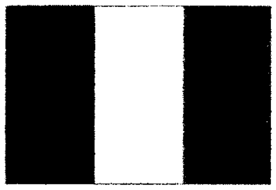
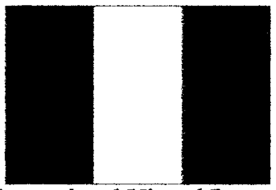
 Base Image	Base Image Define (0 0 256 128) Modify (0 255 0 0 255 255 0 255 0) Virtual Image  Instantiated Virtual Image
Black Blue Green Red White 0.0 0.33 0.0 0.33 0.33	Black Blue Green Red White 0.0 0.0 0.33 0.33 0.33
Color Histogram of Base Image	*Color Histogram of Virtual Image*

Figure 3-4. Colors in a Virtual Image not Present in its Base Image [Flag, 2003]

Figure 3-5 displays the proposed algorithm for processing range queries, which consists of three main tasks. The first task is to identify the desired histogram bin from the given query. This is accomplished in the first three steps of the algorithm, which are the same in both the proposed algorithm and the conventional approach since both need to determine the desired histogram bin. The second task is to determine B_S, the set of binary images that satisfy the query. This can be accomplished using existing query processing techniques such as color histograms ([Hafn, 1995], [Orte, 1998], [Park, 1999]), which are suitable for handling images stored in a binary format. The third task in the algorithm is to determine V_S, the set of virtual images that satisfy the query when they are instantiated. This is accomplished by examining the description of each virtual image and determining the effects that its sequence of editing operations may have on the histogram of its base. Although it may be impossible to identify the exact value contained in a histogram bin, the rules allow the image retrieval system to establish maximum and minimum bounds. To summarize, the proposed algorithm of processing range queries includes techniques to identify minimum and maximum bounds on the values of the histograms bins for virtual images.

```
/* Initialize the parameters of the given query */
1. Initialize the Results set to ∅.
2. Analyze the given query in order to identify the desired query range [$PCT_{min}$,
   $PCT_{max}$] and query color $C_Q$.
3. Use the quantization function (Figure 2-2) on the query color to determine the
   desired Histogram Bin HB.

/* Use histograms to identify the binary images that satisfy the given query */
4. For each histogram tuple extracted from a binary image,
   4.1. If the value in bin HB is within the query range [$PCT_{min}$, $PCT_{max}$],
        4.1.1. Add the image ID in the histogram tuple to the Results set.

/* Use the proposed rules to identify the virtual images that satisfy the given query*/
5. For each virtual image in the database,
   5.1. Execute the Bounds() function to obtain the minimum and maximum bounds
        ($BOUND_{min}$, $BOUND_{max}$) on the percentage of pixels that can be in bin HB.
   5.2. If the range formed by the estimated bounds intersects the query range,
        5.2.1. Add the ID of the virtual image to the Results set.

/* Retrieve the identified images to the user. Any virtual images must first be
instantiated. */
6. Display the images corresponding to the IDs contained in the Results set.
```

Figure 3-5. Proposed Algorithm for Processing Range Queries in a Virtual Image Retrieval System

If the range formed by these minimum and maximum bounds intersects the range [PCT_{min}, PCT_{max}] specified in the query, the proposed algorithm considers that the virtual image satisfies the query. This strategy may lead to retrieving images that do not satisfy the query since it is possible for the query range to intersect the minimum and maximum bounds, but not to completely overlap it. Since the purpose is to retrieve all images that satisfy the query to the user, this research adopts the policy that it is preferable to falsely retrieve an image as opposed to incorrectly omitting one.

The key step in the proposed retrieval algorithm displayed in Figure 3-5 is Step 5.1. The BOUNDS procedure determines if a virtual image satisfies a given query using a set of *rules* that indicate how an image editing operation affects the minimum and maximum bounds on the percentage of pixels that may be of color C_Q in an image. The rules are defined for the set of editing operations Define, Combine, Modify, Mutate, and Merge that were described in Chapter 2. The BOUNDS algorithm is presented in the next section.

3.3. Algorithm for Determining Bounds on Bin HB in a Virtual Image

The algorithm for determining the bounds on bin HB in a virtual image V_i is displayed in Figure 3-6. The goal of the algorithm is to compute the maximum and minimum bounds on the percentage of pixels that may be of color C_Q. The range formed by the maximum and minimum bounds can then be compared to the range requested by the query, which will indicate if V_i satisfies the query. For example, if the algorithm determines that no more than 50% of the virtual image is in the bin corresponding to white (RGB color [255, 255, 255]), the system knows that V_i cannot satisfy the query "*Retrieve all images that are between 50% and 100% color [255, 255, 255]*".

```
BOUNDS
/* Use the base image of $V_i$ to initialize the [$BOUND_{min}$, $BOUND_{max}$] range, which is the
number of pixels that can be in bin HB, and imageSize, which is the total number of pixels in
$V_i$ */
1. Let $B_i$ represent the base image of $V_i$
2. Let $H_i$ represent the histogram corresponding to base image $B_i$
3. Initialize ImageSize as the number of pixels in base image $B_i$
4. Initialize $BOUND_{max}$ as ImageSize $\times$ the value in bin HB of histogram $H_i$
5. Initialize $BOUND_{min}$ to the same value as $BOUND_{max}$

/* Sequentially access each operation in the virtual image */
6. For each operation OP in $V_i$

    /* The Define Operation does not change an image, so it does not alter the [$BOUND_{min}$,
    $BOUND_{max}$] range or imageSize. Instead, the algorithm must compute and track the
    number of pixels in the DR. */
    6.1. If OP is Define($x_1$, $y_1$, $x_2$, $y_2$)
        6.1.1. Let DR_Size = $(|x_1 - x_2|) \times (|y_1 - y_2|)$, which is the number of pixels in the
               Defined Rectangle.
        6.1.2. Execute the NEW LIMIT algorithm to compute limits on the number of
               pixels that may change before the next Define operation.

    /* Update the [$BOUND_{min}$, $BOUND_{max}$] range and imageSize variables for each
    operation other than Define. */
    6.2. Else
        6.2.1. Compute $BOUND_{max}$ using the proposed rules for operation OP
        6.2.2. Compute $BOUND_{min}$ using the proposed rules for operation OP
        6.2.3. Compute ImageSize using the proposed rules for operation OP

        /* Check if operation OP can change more pixels than the limits computed
        previously. */
        6.2.4. If New Limit column for OP is "Yes"
            6.2.4.1. Execute NEW LIMIT to compute new limits on the number of pixels
                     that may change before the next Define operation

/* Convert the boundary range on the number of pixels in bin HB to the percentage of pixels
in bin HB. */
7. $BOUND_{max}$ = $BOUND_{max}$ / ImageSize
8. $BOUND_{min}$ = $BOUND_{min}$ / ImageSize


NEW LIMIT
/* Some operations only affect the DR. Two such operations in succession will still only
affect the DR. The purpose of this method is to compute limits on the number of pixels that
may change after applying the above operations in succession. Note: The variables $Temp_{min}$
and $Temp_{max}$ are used within the proposed rules. */
1. $Temp_{min}$ = MAX($BOUND_{min}$ – DR_Size, 0)
2. $Temp_{max}$ = MIN($BOUND_{max}$ + DR_Size, ImageSize)
```
Figure 3-6. Algorithm for Determining Bounds on bin HB in a Virtual Image

The BOUNDS algorithm computes the bounds on the percentage of pixels in bin HB by examining each operation in the description of the current virtual image. So, it iteratively computes new values for the bounds for each operation in the description. The algorithm performs this computation using rules that will be presented in the next section. The remainder of this section describes the purpose of each step in the BOUNDS algorithm in more detail.

Step 1 of the algorithm is to identify the base B_i of the virtual image. To identify B_i, the system must read the description of V_i and access the referenced base image that is contained in the first line of the description. If this base image is itself a virtual image, then the system must recursively read its description to obtain its base image. This process is repeated until a binary base image is identified. The identification of the base image leads directly to Step 2, which identifies the histogram of the base image.

Steps 3, 4, and 5 initialize the variables that will continually be updated by the proposed rules during the execution of the BOUNDS algorithm. Specifically, Step 3 of the algorithm computes imageSize to be the size of the base image B_i, where the size of an image is defined as the number of pixels it contains. The proposed algorithm, then, requires that the sizes of the binary images are stored in the database. Steps 4 and 5 initialize $BOUND_{min}$ and $BOUND_{max}$ to the number of pixels that are in bin HB in the base image. This bound is computed by accessing the histogram corresponding to B_i and multiplying the value in the bin HB by imageSize. Although they are initialized to the same value, $BOUND_{min}$ and $BOUND_{max}$ may spread apart as the algorithm proceeds.

With the bounds initialized, Step 6 of the algorithm contains a loop that accesses the editing operations in the description of the virtual image in order. When a Define operation is encountered, the system knows that the subsequent operations will act on the rectangle specified in the parameters of the operation. The Modify and Combine operations only affect pixels inside the Defined Rectangle (DR), which means that successive applications of these operations can only change those pixels. Consequently, the algorithm computes the number of pixels contained in the DR and uses it to compute minimum and maximum limits on the numbers of pixels that may change until the next Define operation in the virtual image description is encountered. Thus, the maximum number of pixels that can change after successive Modify and Combine operations is equal to DR_Size. Consequently, the algorithm computes the limit on the minimum bound after the application of successive Modify and Combine operations as $\text{BOUND}_{\text{min}}$ – DR_Size, and stores this value in the variable Temp_{min}. Similarly, the algorithm computes the limit on the maximum bound as $\text{BOUND}_{\text{max}}$ + DR_Size and stores this value in the variable Temp_{max}. These computations are performed in the NEW LIMIT procedure.

With the exception of the Define operation, the editing operations in ([Grue, 1996], [Spee, 1998], [Spee, 2000]) make changes to the pixels in a virtual image. So, when an operation other than Define is encountered in the BOUNDS algorithm, the algorithm adjusts the maximum and minimum bounds on the number of pixels in bin HB in Steps 6.2.1 and 6.2.2 as well as the variable imageSize in Step 6.2.3. The adjustments are made based on the rules defined for each operation that will be presented in the next section. In addition, the BOUNDS algorithm checks each operation in Step 6.2.4. to determine if it can affect pixels outside the current DR. If so, it executes the procedure NEW LIMIT to update the limit variables Temp_{min} and Temp_{max}.

After the algorithm has accessed all of the editing operations in V_i, it has its final minimum and maximum bounds on the number of pixels that are in bin HB. Steps 7 and 8 convert these values to percentages by dividing them by the total number of pixels in the image, which is in imageSize. The BOUNDS algorithm returns these values back to the algorithm in Figure 3-5, so that it can utilize these percentages to determine if the resulting virtual image could satisfy the given range query.

3.4. Derivation of Bounds

As described in the previous section, the proposed algorithm continually adjusts the image size and the maximum and minimum bounds on the number of pixels that are in bin HB based on the editing operations listed in the description of the virtual image. These adjustments are listed in Table 3-2. The bounds for the editing operations are dependent on their associated parameters, so the "Parameters Conditions" column in the table describes the conditions that the parameters must meet to apply the corresponding bounds. The "Update Limit" column of the table is used to indicate which operations can affect the pixels outside of the Defined Rectangle and will therefore need to update the limit variables $Temp_{min}$ and $Temp_{max}$. The following sections explain how each of the bounds is derived.

Editing Operation	Parameters Conditions	Minimum Bound	Maximum Bound	Image Size	New Limit
Combine	All	$BOUND_{min}$	$BOUND_{max}$	ImageSize	No
Modify	HB $\notin$ Quantize (R_{new}, G_{new}, B_{new}) AND HB $\notin$ Quantize (R_{min}:R_{max}, G_{min}:G_{max}, B_{min}:B_{max})	$BOUND_{min}$	$BOUND_{max}$	ImageSize	No
	HB $\notin$ Quantize (R_{new}, G_{new}, B_{new}) AND HB $\in$ Quantize (R_{min}:R_{max}, G_{min}:G_{max}, B_{min}:B_{max})	MAX[$Temp_{min}$, α] where α=MAX(0, $BOUND_{min}$– DR_Size)	$BOUND_{max}$	ImageSize	No
	HB $\in$ Quantize (R_{new}, G_{new}, B_{new}) AND HB $\in$ Quantize (R_{min}:R_{max}, G_{min}:G_{max}, B_{min}:B_{max})	$BOUND_{min}$	MIN[$Temp_{max}$, α] where α=MIN(ImageSize, $BOUND_{min}$+ DR_Size)	ImageSize	No
	HB $\in$ Quantize (R_{new}, G_{new}, B_{new}) AND HB $\notin$ Quantize (R_{min}:R_{max}, G_{min}:G_{max}, B_{min}:B_{max})	$BOUND_{min}$	MIN[$Temp_{max}$, α] where α=MIN(ImageSize, $BOUND_{min}$+ DR_Size)	ImageSize	No
Mutate	ImageSize == DR_Size	$BOUND_{min}\times\|M_{11}\times M_{22}\|$	$BOUND_{max}\times\|M_{11}\times M_{22}\|$	ImageSize $\times\|M_{11}\times M_{22}\|$	Yes
	Rigid Body	MAX[$Temp_{min}$, α] where α=MAX(0, $BOUND_{min}$– DR_Size)	MIN[$Temp_{max}$, α] where α=MIN(ImageSize, $BOUND_{max}$+ DR_Size)	ImageSize	No
	Otherwise	$BOUND_{min}$	$BOUND_{max}$	ImageSize	No
Merge	Target is NULL	MAX[0, DR_Size – (ImageSize – $BOUND_{min}$)]	MIN[$BOUND_{max}$, DR_Size]	DR_Size	Yes
	Target is Not NULL	MAX[0, DR_Size – (ImageSize – $BOUND_{min}$)] + MAX(0, TargetHB – DR_Size)	MIN($BOUND_{max}$, DR_Size) + MIN(TargetHB, MAX(0, TargetSize – DR_Size))	[MAX((x_p+x_2–x_1), height) – MIN(x_p,0)+1] $\times$ [MAX((y_p+y_2–y_1), width) – MIN(y_p,0)+1]	Yes

Table 3-2 – Rules For How Editing Operations Affect Bounds on Histogram Bins

3.4.1. Combine (C_{11}, C_{12}, C_{13}, C_{21}, C_{22}, C_{23}, C_{31}, C_{32}, C_{33})

The Combine operation changes the intensity values of only the pixels in the Defined Rectangle using the weighted average of the intensity values of the pixels' neighbors. Since the operation only affects the pixels in the DR, the BOUNDS algorithm does not need to update the temporary limit variables $Temp_{min}$ and $Temp_{max}$. The parameters to the Combine operation are the weights used in computing the weighted average. This definition of the Combine operation leads to the following observations:

1. The Combine operation does not add or remove pixels from an image.
2. If a pixel has the same intensity value as all of its neighbors, it will not be changed as a result of the Combine operation.

To derive the rule governing how the Combine operation affects the size of an image, meaning the number of pixels it contains, consider observation 1. Since no new pixels are added or removed from an image, the number of pixels must stay the same. Consequently, one rule for the effect of the Combine operation is that the image size remains the same.

To derive the maximum and minimum bounds on the value in bin HB, consider observation 2. This observation implies that if there is a homogenous region inside an image, only the pixels on the boundary of the region can change color. Because of this operation, the algorithm assumes that the number of pixels in bin HB will remain the same after the application of the Combine operation. Consequently, two more rules for the effect of the Combine operation are that the minimum and maximum bounds on the value in bin HB remain the same.

To summarize, let the number of pixels in an image be imageSize, the minimum bound on the value in bin HB be $\text{BOUND}_{\text{min}}$, and the maximum bound on the value in bin HB be $\text{BOUND}_{\text{max}}$. The expressions for these values after applying a Combine operation according to the proposed rules are:

- Size of image: imageSize
- Minimum Bound on HB: $\text{BOUND}_{\text{min}}$
- Maximum Bound on HB: $\text{BOUND}_{\text{max}}$

3.4.2. Modify (R_{min}, R_{max}, R_{new}, G_{min}, G_{max}, G_{new}, B_{min}, B_{max}, B_{new})

Like the Combine operation, the Modify operation changes the intensity values of some of the pixels in the Defined Rectangle. Also like the Combine operation, the BOUNDS algorithm does not need to update the temporary limit variables Temp_{min} and Temp_{max} since the operation can only affect the pixels in the DR. The parameters of the operation specify which pixels get changed and what their new intensity values will become. This leads to the following observations:

3. The Modify operation does not add or remove pixels from an image.
4. The pixels that will change colors as a result of the operation are specified in the parameters.
5. Since only the pixels that are in the Defined Rectangle can change, the maximum number of pixels that can change as a result of this operation is equal to the size of the Defined Rectangle. This value is stored in DR_Size.

Observation 3 is similar to Observation 1. So, one rule for the Modify Operation is very similar to a rule for the Combine operation. Specifically, since no new pixels are added or removed as a result of the Modify operation, one of its rules specifies that the image size remains the same.

To derive the rules for how the Modify operation affects the bounds on the number of pixels within bin HB, it is necessary to consider Observations 4 and 5. Observation 5 indicates that the bound on the number of pixels that can change color is equal to DR_Size. This means that if there are x pixels in bin HB before the application of the Modify operation, then there will be at most x + DR_Size pixels and no fewer than x – DR_Size pixels in the bin after its application.

Observation 4 indicates that these bounds may be improved by examining the parameters of the operation. In the operation Modify (R_{min}, R_{max}, R_{new}, G_{min}, G_{max}, G_{new}, B_{min}, B_{max}, B_{new}), the only pixels that change are ones with Red values in the range [R_{min}, R_{max}], Green values in the range [G_{min}, G_{max}], and Blue values in the range [B_{min}, B_{max}]. These pixels are changed to the color (R_{new}, G_{new}, B_{new}). So, the rules for determining the effects of the Modify operation should examine the parameters to identify the colors of the pixels in the DR that change and the new color that the pixels will become. Note that since the query is based on the number of pixels in histogram bins that represent quantized colors, the colors in the parameters of the Modify operation must be quantized.

Consider when no colors with a Red intensity value within [R_{min}, R_{max}], a Green intensity value within [G_{min}, G_{max}], and a Blue intensity value within [B_{min}, B_{max}] quantize to bin HB. This means that no pixels which map to bin HB will change in the image, so the value in bin HB will not decrease. Based on this information, one rule is that if Quantize(R_x, G_x, B_x) $\neq$ HB for all $\{R_x, G_x, B_x \mid R_x \in [R_{min}, :R_{max}], G_X \in [G_{min}, G_{max}], B_X \in [B_{min}, B_{max}]\}$, then the minimum bound on HB stays the same.

Now consider when Quantize(R_{new}, G_{new}, B_{new}) equals HB. This means that the pixels within the Defined Rectangle may be changed to a color that quantizes to bin HB. Since there are DR_Size pixels in the Defined Rectangle, the maximum bound on HB may increase by DR_Size. Note that this also implies that the minimum bound on HB should stay the same.

Next, consider when Quantize(R_{new}, G_{new}, B_{new}) does not equal HB. This means no pixels in the Defined Rectangle will be changed to a color that quantizes to bin HB. So, the maximum bound on HB should stay the same. However, pixels whose colors do quantize to bin HB may change their colors if their Red intensity values are in the range [R_{min},:R_{max}], Green intensity values are in the range [G_{min}, G_{max}], and Blue intensity values are in the range [B_{min}, B_{max}]. In such a case, the minimum bound on HB should decrease by DR_Size.

To summarize, the rules for the Modify operation are dependent on its parameters. As with the Combine operation, let the number of pixels in an image be imageSize, the minimum bound on the value in bin HB be $BOUND_{min}$, and the maximum bound on the value in bin HB be $BOUND_{max}$. The expressions for these values after applying a Modify operation according to the proposed rules are:

If (Quantize(R_{new}, G_{new}, B_{new}) == HB), then
- Size of image: imageSize
- Minimum Bound on HB: $BOUND_{min}$
- Maximum Bound on HB: $BOUND_{max}$ + DR_Size

If (Quantize(R_{new}, G_{new}, B_{new}) $\neq$ HB) and
(Quantize(R_x, G_x, B_x) $\neq$ HB for all {R_x, G_x, B_x | $R_x \in [R_{min},:R_{max}]$, $Gx \in [G_{min}, G_{max}]$, $Bx \in [B_{min}, B_{max}]$}), then
- Size of image: imageSize
- Minimum Bound on HB: $BOUND_{min}$
- Maximum Bound on HB: $BOUND_{max}$

If (Quantize(R_{new}, G_{new}, B_{new}) $\neq$ HB) and

$\exists$ {R_x, G_x, B_x | $R_x \in [R_{min},:R_{max}]$, $G_x \in [G_{min}, G_{max}]$, $B_x \in [B_{min}, B_{max}]$ and (Quantize(R_x, G_x, B_x) == HB)}, then

- Size of image: imageSize
- Minimum Bound on HB: $BOUND_{min}$ – DR_Size
- Maximum Bound on HB: $BOUND_{max}$

3.4.3. Mutate (M_{11}, M_{12}, M_{13}, M_{21}, M_{22}, M_{23}, M_{31}, M_{32}, M_{33})

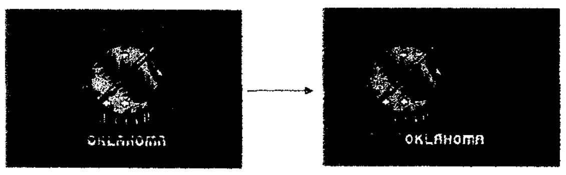
The Mutate operation changes the location of the pixels in the Defined Rectangle. By moving a pixel from one area in an image to another area, the Mutate operation has the ability to overwrite pixels in other areas of the image. In addition, it can move the pixels to new locations outside the current image. This information leads to the following observations:

6. Since the Mutate Operation can overwrite pixels in various areas of an image, it can change pixels outside of the Defined Rectangle.
7. Since the Mutate Operation can move pixels to new locations, it has the capability to enlarge or shrink an image. Thus, it can alter the number of pixels that are contained within an image.

Because pixels outside the DR can change as indicated in Observation 6, it is difficult defining rules for the Mutate operation. Observation 7 indicates that the effects of the operation can vary based on its parameters. Thus, the algorithm assumes that the effects of most Mutate operations are unknown and consequently, does not update the bound variables.

The rules that are defined in the BOUNDS algorithm for the Mutate operation are for specific conditions of its associated parameters. One such rule occurs when the Defined Rectangle contains the entire image. Since all of the pixels are affected in this situation, the distribution of colors in the image should remain constant, except for extreme cases such as shrinking an image down to one pixel. A rule then is that when the entire image is in the Defined Rectangle, the minimum bound for HB, maximum bound for HB, and image size should be changed by the scaling factors of the mutation matrix (M_{11} and M_{22}). In addition, the temporary limit variables $Temp_{min}$ and $Temp_{max}$ should be updated since pixels outside the DR have been changed.

Another rule occurs when the mutation matrix is a rigid body transformation, meaning that it consists of only translations and rotations. In this situation, the transformation moves one section of an image to another area in the image. Although possible, it is assumed the image size stays constant. The original location of the pixels will be changed to some default color, currently RGB value (0, 0, 0), as displayed in Figure 3-7.



Oklahoma Flag with Defined Rectangle

Flag after Application of Mutate (1, 0, 0, 0, 1, -80, 0, 0, 0, 1)

Figure 3-7. Results after Application of a Mutate Operation that Translates the DR

To develop a rule for the above situation, consider a pixel in the Defined Rectangle. The pixel retains its intensity value when it is moved to another location in the image. The pixel at the new location is lost, while a pixel with the color (0, 0, 0) is added. So, the effect of the operation is that the histogram bin corresponding to (0, 0, 0) may gain pixels while all the other histogram bins may lose pixels. The maximum number of pixels that are replaced is equal to the number of pixels in the Defined Rectangle. So, the rule for this situation is that both the minimum and maximum bounds may be altered by the value in DR_Size, and the total number of pixels in the image will not change. Also, since the maximum number of pixels that are replaced is equal to the number of pixels in the DR, the temporary limit variables $Temp_{min}$ and $Temp_{max}$ do not have to be updated.

To summarize, the algorithm assumes that the effects of the Mutate operation are unknown and therefore, does not update the bounds and image size. The rules that are defined for the Mutate operation are dependent on its parameters like the Modify operation. Given the same variable definitions as in the previous operations, the new values of the bounds and image size are:

If (DR contains entire image), then

- Size of image: imageSize × (M_{11} × M_{22})
- Minimum Bound on HB: $BOUND_{min}$ × (M_{11} × M_{22})
- Maximum Bound on HB: $BOUND_{max}$ × (M_{11} × M_{22})

If the transformation is a rigid body transformation, then

- Size of image: imageSize
- Minimum Bound on HB: $BOUND_{min}$ – DR_Size
- Maximum Bound on HB: $BOUND_{max}$ + DR_Size

3.4.4. Merge (Target Image, x_P, y_P)

The Merge operation combines the Defined Rectangle of the base image and the Target Image. If the Target Image is NULL, then the Merge operation simply crops the Defined Rectangle out of the current image. Thus, the Merge operation affects all of the pixels outside of the Defined Rectangle. Since pixels outside the DR can be affected, the BOUNDS algorithm must update the values of $Temp_{min}$ and $Temp_{max}$.

Since the Merge operation combines two images, the following observations can be made:

8. To identify the minimum number of pixels possible in bin HB after applying the Merge operation, it is necessary to identify the minimum number of pixels that could possibly be in the DR and in the Target Image.
9. To identify the maximum number of pixels possible in bin HB after applying the Merge operation, it is necessary to identify the maximum number of pixels that could possibly be in the DR and in the Target Image.
10. The size of the resulting image will be equal to the size of the Target Image, unless the DR is copied onto a position that causes the image to grow.
11. If the Target Image is NULL, the size of the resulting image will be equal to the size of the DR.

Observations 8 and 9 indicate that is necessary to identify the minimum and maximum possible number of pixels in bin HB contained within the DR and Target Image in order to identify the new bounds on that value. Observation 10 indicates that is necessary to identify where a DR is pasted in a Target Image to determine its size. Finally, Observation 11 indicates that it is necessary to check if the Target Image is NULL when developing the rules.

Depending on where the DR is located in the image, the number of pixels in it that are in bin HB can fluctuate. For example, consider that the amount of white pixels in a DR specified in an image is half red and half white as in Figure 3-8. If the DR is small, as in Figures 3-8A and 3-8B, it is possible that it may contain all white pixels or no white pixels. So, the number of white pixels in it could be anywhere from DR_SIZE to zero. Alternatively, if the DR is large, it would be impossible for it to contain only white pixels as in Figure 3-8C, and it would be impossible for it to contain no white pixels as in Figure 3-8D.

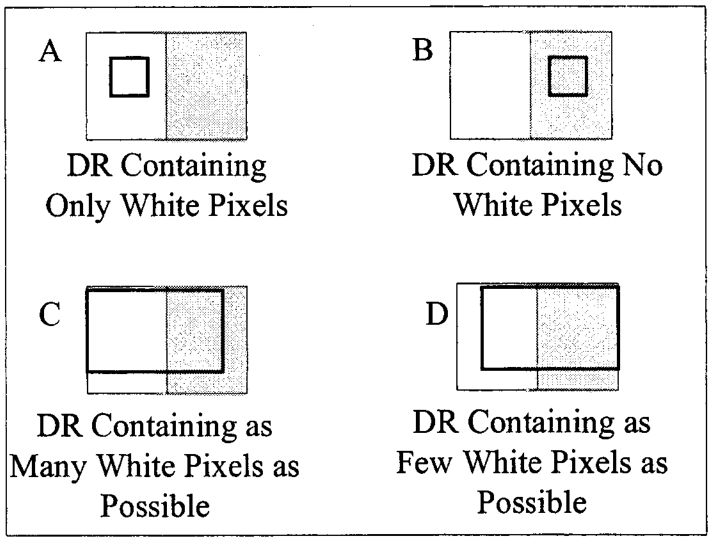

A
DR Containing Only White Pixels
B
DR Containing No White Pixels
C
DR Containing as Many White Pixels as Possible
D
DR Containing as Few White Pixels as Possible

Figure 3-8. Defined Rectangles that have Varying Numbers of White Pixels

The formulae for determining the minimum and maximum number of pixels in bin HB within a Defined Rectangle, then, are dependent upon the size of the DR and the number of pixels in bin HB within the base image. The minimum number of pixels is equal to the number of pixels in the DR minus the number of pixels in the image that are not in bin HB. Using the above variables, this value is computed as DR_Size – (ImageSize – $\text{BOUND}_{\text{min}}$). The maximum number of pixels in bin HB within a DR is equal to the minimum of the number of pixels in the DR and the number of pixels in bin

HB in the entire image. Using the above variables, this value is MIN(BOUND_{max}, DR_Size).

Similar reasoning can be used to compute the formulae for the minimum and maximum numbers of pixels in bin HB within the Target Image. The difference is that the formulae must compute how many of the pixels in bin HB within the target are covered by the DR. Again, this will be dependent upon the size of the DR and the number of pixels in bin HB within the Target Image. Let TargetSize represent the number of pixels in the Target Image and TargetHB represent the number of pixels in bin HB within the Target Image. The minimum number of pixels in bin HB within the Target Image after the application of the Merge operation is equal to the number of pixels in the DR subtracted from the number of pixels in bin HB in the Target Image before the operation. Using the above variables, this expression is TargetHB – DR_Size. The maximum number of pixels in bin HB in the Target Image after a Merge operation is equal to the minimum of the number of pixels in the Target Image before the operation and the number of pixels in the Target Image not covered by the Defined Rectangle. This value can be expressed as MIN(TargetHB, TargetSize – DR_Size). Since each of the above expressions represents the number of pixels that are mapped to bin HB, the value cannot be negative. Thus, each of the expressions should be bounded below by zero.

In a Merge operation, the DR is copied into the Target Image, so the size of the resulting image will be equal to the size of the TargetImage. The only exceptions occur when the DR is copied into a position that will make the resulting image larger than the target. One such example is illustrated in Figure 3-9 where the DR is pasted in the lower right corner of the Target Image. Figure 3-9A displays a Target Image. Figure 3-9B displays a DR being copied onto the lower right corner of the Target Image. Figure 3-9C displays that the resulting image must add pixels in order for the resulting image to be rectangular.

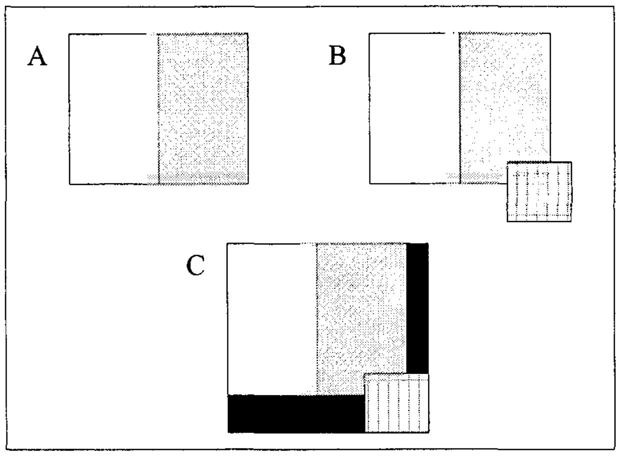


Figure 3-9. Results of a Merge Operation Larger than the Target Image

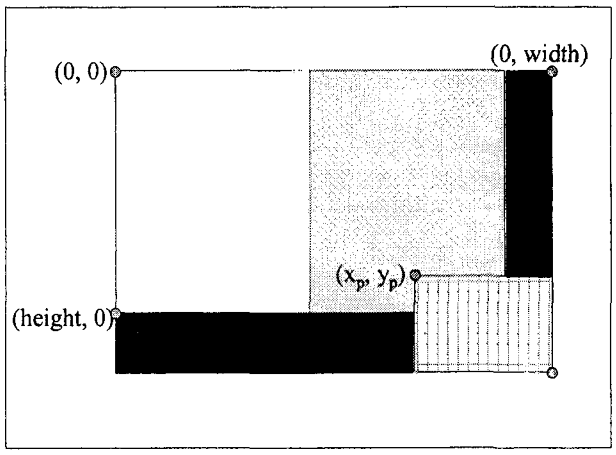


Figure 3-10. Coordinates in an Image Resulting from Merge (Target, x_p, y_p)

Let height represent the number of rows in the Target Image and width represent the number of columns. This information will have to be stored in the database. The position that the DR will be copied into the Target Image is given in the parameters of the

Merge operation. To find the size of the resulting image, it is necessary to find the coordinates of the resulting image as in Figure 3-10.

If the image grows as in Figure 3-10, the new lower right coordinates would be (x_p + the height of the DR, y_p + the width of the DR). The height and width of the DR can be computed from the parameters of the Define operation, which should still be in memory from Step 6.1 of the BOUNDS algorithm. The height is $(x_2 - x_1)$, and the width is $(y_2 - y_1)$. So, the new lower right coordinates of the image would be $(x_p+x_2 - x_1, y_p+y_2 - y_1)$. Alternatively, if the image grows as in Figure 3-11, the new upper left coordinates of the image would simply be (x_p, y_p).

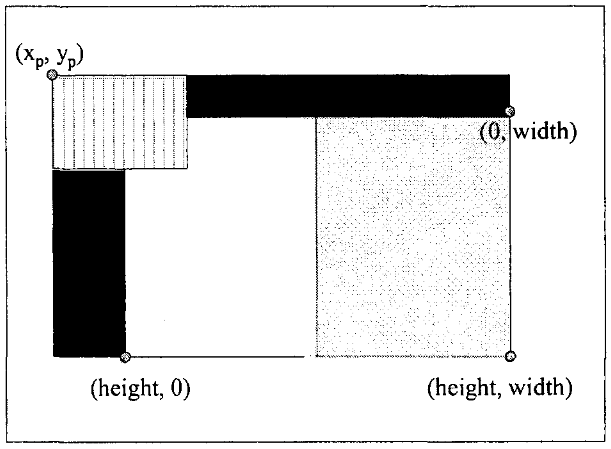


Figure 3-11. Other Possible Coordinates Resulting from Merge (Target, x_p, y_p)

Using the above information, the size of the resulting image would be the product of the height and width of the resulting virtual image. Using the above parameters of the Merge operation, these values are expressed as:

Height: $\text{MAX}((x_p + x_2 - x_1), \text{height of target}) - \text{MIN}(x_p, 0) + 1$.
Width: $\text{MAX}((y_p + y_2 - y_1), \text{width of target}) - \text{MIN}(y_p, 0) + 1$.

To summarize, the rules for the Merge operation are dependent on its parameters. Given the same variable definitions as in the previous operations, the new values of the bounds and image size are:

If target is NULL, then

- Size of image: DR_Size
- Min Bound on HB: MAX(DR_Size – (ImageSize – BOUND_{min}), 0)
- Max Bound on HB: MIN(BOUND_{max}, DR_Size)

If target is not NULL, then

- Size of image: (MAX(($x_p + x_2 - x_1$), height of target) – MIN(x_p, 0) + 1) × (MAX(($y_p + y_2 - y_1$), width of target) – MIN(y_p, 0) + 1)
- Min Bound on HB: (MAX(DR_Size – (ImageSize – BOUND_{min}), 0)) + (MAX(TargetHB – DR_Size, 0))
- Max Bound on HB: (MIN(BOUND_{max}, DR_Size)) + MIN(TargetHB, MAX((TargetSize – DR_Size), 0))

3.5. Range Query Processing Example

This section provides a specific example of using the algorithm in Figure 3-6 on a set of virtual images. For example, consider a system that uses the RGB color model, and uniformly divides each axis in half when quantizing the color space. So, in the quantization function presented in Chapter 2, $(d_1, d_2, d_3) = (2, 2, 2)$ and $(CM_1, CM_2, CM_3) = (256, 256, 256)$. This also means that each color histogram extracted from a binary image will have a total of $2^3 = 8$ bins.

The underlying database contains 4 binary images called B1, B2, B3, and B4, and 4 edited images, called V5, V6, V7, and V8, meaning that the database has 8 total images. Based on this quantization scheme, Table 3-3 contains the four histograms extracted from the binary images in the example database along with the identifiers to their associated images. Table 3-4 contains the descriptions of the 4 virtual images. In addition, the database stores that each binary image contains 100 pixels arranged as 10 rows and 10 columns.

The specific range query used has values of 50 for PCT_{min}, 100 for PCT_{max}, and (255, 255, 255) for color C_Q, which is the color white. This means that the example range query is equivalent to "*Retrieve all images that are at least 50% white*". Using the parameters of the quantization function described earlier, color C_Q maps to histogram bin 7, so HB equals 7.

Histogram ID	**Image ID**	**Bin_0**	**Bin_1**	**Bin_2**	**Bin_3**	**Bin_4**	**Bin_5**	**Bin_6**	**Bin_7**
H1	B1	0	0	0	0	0	0	0.6	0.4
H2	B2	0.5	0.5	0	0	0	0	0	0
H3	B3	0.8	0	0	0	0	0	0	0.2
H4	B4	0	0	0	0	0	0	0.4	0.6

Table 3-3. Histograms for the Binary Images in the Example Database

V5: B2 Define (0, 0, 9, 4) Merge (NULL, 0, 0)	**V6:** B3 Define (0, 0, 4, 3) Modify (0, 100, 255, 0, 100, 255, 0, 100, 255) Combine (1, 2, 1, 2, 4, 2, 1, 2, 1)
V7: B4 Define (0, 0, 9, 0) Mutate (1, 0, 0, 0, 4, 0, 0, 0, 1)	**V8:** B4 Define (0, 0, 9, 2) Modify (0, 255, 255, 0, 255, 255, 0, 255, 255)

Table 3-4. Descriptions of Virtual Images in the Example Database

V5: *B2*
Define (0, 0, 9, 4)
Merge (NULL, 0, 0)

When the BOUNDS algorithm is applied to V5, the first two steps of the algorithm identify B2 as the base image and H2 as its histogram. The next three steps identify imageSize, $BOUND_{max}$, and $BOUND_{min}$. The imageSize variable is set to 100, since there are 100 pixels in B2. Both $BOUND_{max}$ and $BOUND_{min}$ are set to the product of the value in bin 7 in the histogram of B2 and imageSize. Since bin 7 has a value of 0, this product is also 0.

Step 6 processes the editing operations in V5. Since the first operation is Define (0, 0, 9, 4), the variable DR_Size is set to the number of pixels in the DR, which is $(|0 - 9| + 1) \times (|0 - 4| + 1) = 50$. In addition, Temp_{min} is set to 0, and Temp_{max} is set to 50.

The algorithm then applies the rules in Table 3-2 for the second operation Merge (NULL, 0, 0). This operation crops 50 pixels from the base image. Applying the rules in Table 3-2 for this operation, the algorithm sets $\text{BOUND}_{\text{min}}$ to MAX[0, 50 – (100-0)], which equals 0, and sets $\text{BOUND}_{\text{max}}$ to MIN[0, 50], which also equals 0. Next, the algorithm sets the imageSize variable equal to DR_Size, which is 50. Finally, the Merge operation does affect pixels outside the DR, so the algorithm updates the temporary minimum temp_{max} to MAX(0 – 50, 0) = 0 and the temporary maximum temp_{max} to MIN (0 + 50, 50) = 50.

The last two steps convert $\text{BOUND}_{\text{min}}$ and $\text{BOUND}_{\text{max}}$ to percentages by dividing them by imageSize. So, $\text{BOUND}_{\text{min}}$ equals 0 / 50 = 0, and $\text{BOUND}_{\text{max}}$ equals 0 / 50 = 0. The result of the BOUNDS algorithm, then, is that the percentage of pixels that are white in V5 will be in the range [0, 0], which means that it will not have any white pixels. This is to be expected since the description of V5 implies that it is created by cropping 50 of the pixels in image B2, and B2 does not have any white pixels. Since the range [0, 0] does not intersect the query range [.5, 1], V5 would not be returned to the user.

V6: B3

Define (0, 0, 4, 3)

Modify (0, 100, 255, 0, 100, 255, 0, 100, 255)

Combine (1, 2, 1, 2, 4, 2, 1, 2, 1)

When the BOUNDS algorithm is applied to V6 for the same query, the first two steps of the algorithm identify B3 as the base image and H3 as its histogram. The imageSize variable is set to 100, since there are 100 pixels in B3. The next two steps set $BOUND_{max}$ and $BOUND_{min}$ to the product of bin 7 in the histogram of B2 and imageSize. Since bin 7 in Histogram H3 has a value of 0.2, the initial values of both $BOUND_{min}$ and $BOUND_{max}$ are 20.

Step 6 processes the editing operations in V6. Since the first operation is Define (0, 0, 4, 3), the variable DR_Size is set to the number of pixels in the DR, which is $(|0 - 4| + 1) \times (|0 - 3| + 1) = 20$. Since the subsequent editing operations until the next Define operation may only alter these 20 pixels, the limit $Temp_{min}$ is set to $(BOUND_{min} - 20)$, which is 0, and $Temp_{max}$ is set to $(BOUND_{max} + 20)$, which is 40.

Next, the algorithm applies the rules in Table 3-2 for the second operation Modify (0, 100, 255, 0, 100, 255, 0, 100, 255). This operation converts the pixels with intensify values in between 0 and 100 in all three color axes to white, the query color. So, this operation may create new white pixels. As a result, the rules in Table 3-2 do not change the values of $BOUND_{min}$ and imageSize, so they remain equal to 20 and 100, respectively. The rules do compute a new value for $BOUND_{max}$, which is equal to MIN[40, MIN(100, 20 + 20)] = 40. The Modify operation does not change pixels outside the DR, so $Temp_{min}$ and $Temp_{max}$ are not updated.

For the third operation, Combine (1, 2, 1, 2, 4, 2, 1, 2, 1), the rules in Table 3-2 do not change the values of $\text{BOUND}_{\text{min}}$, $\text{BOUND}_{\text{min}}$, or imageSize. Consequently, they remain equal to 20, 40, and 100, respectively. As before, the Combine operation does not change pixels outside the DR, so Temp_{min} and Temp_{max} are not changed.

When the algorithm finishes processing all of the operations, the last two steps convert $\text{BOUND}_{\text{min}}$ and $\text{BOUND}_{\text{max}}$ to percentages of imageSize. So, $\text{BOUND}_{\text{min}}$ equals 20 / 100 = 0.2, and $\text{BOUND}_{\text{max}}$ equals 40 / 100 = 0.4. The result of the BOUNDS algorithm, then, is that the percentage of pixels that are white in V6 will be in the range [.2, .4]. Since this range does not intersect the query range [.5, 1], V6 would not be returned to the user.

V7: B4
Define (0, 0, 9, 0)
Mutate (1, 0, 0, 0, 4, 0, 0, 0, 1)

When the BOUNDS algorithm is applied to V7 for the same query, the first two steps of the algorithm identify B4 as the base image and H4 as its histogram. The imageSize variable is set to 100, since there are 100 pixels in B4. The next two steps set $\text{BOUND}_{\text{max}}$ and $\text{BOUND}_{\text{min}}$ to the product of bin 7 in the histogram of B4 and imageSize. Since bin 7 in Histogram H4 has a value of 0.6, both $\text{BOUND}_{\text{min}}$ and $\text{BOUND}_{\text{max}}$ are set to 60.

Since the first operation is V7 is Define (0, 0, 9, 0), the variable DR_Size is set to the number of pixels in the DR, which is $(|0 - 9| + 1) \times (|0 - 0| + 1) = 10$. So, the next set of operations until the next Define operation will alter these 10 pixels. Consequently, the limit $Temp_{min}$ is set to ($BOUND_{min}$ – 10), which is 50, and $Temp_{max}$ is set to ($BOUND_{max}$ + 10), which is 70.

Next, the algorithm applies the rules in Table 3-2 for the second operation Mutate (1, 0, 0, 0, 4, 0, 0, 0, 1). This operation enlarges the DR by a factor of 4 in the y-direction. The system does not have enough information to know which pixels are increased, so the rules in Table 3-2 do not change the values of $BOUND_{min}$, $BOUND_{max}$, or imageSize. Thus, they remain 60, 60, and 100, respectively. Since the effect of the Mutate operation is unknown, the algorithm does not change $Temp_{min}$ and $Temp_{max}$.

The last two steps of the algorithm convert $BOUND_{min}$ and $BOUND_{max}$ to percentages of imageSize. So, both bounds are set equal to 60 / 100 = 0.6. The result of the BOUNDS algorithm, then, is that the percentage of pixels that are white in V6 will be in the range [.6, .6]. Since this range does intersect the query range [.5, 1], V7 is returned to the user

V8: B4
Define (0, 0, 9, 2)
Modify (0, 255, 255, 0, 255, 255, 0, 255, 255)

Since V8 has the same base image as V7, the first five steps of the algorithm will produce the same values. So, as in V7, imageSize, $BOUND_{max}$, and $BOUND_{min}$ will equal 100, 60, and 60, respectively.

Since the first operation is V7 is Define (0, 0, 9, 2), the variable DR_Size is set to the number of pixels in the DR, which is $(|0 - 9| + 1) \times (|0 - 2| + 1) = 30$. In addition,

$Temp_{min}$ is set to ($BOUND_{min}$ – 30), which is 30, and $Temp_{max}$ is set to ($BOUND_{max}$ + 30), which is 90.

Next, the algorithm applies the rules in Table 3-2 for the second operation Modify (0, 255, 255, 0, 255, 255, 0, 255, 255). This operation changes all of the pixels in the DR to white. As a result, the rules in Table 3-2 do not change the values of $BOUND_{min}$ and imageSize, so they remain equal to 60 and 100, respectively. The rules do compute a new value for $BOUND_{max}$, which is equal to MIN[90, MIN(100, 60 + 30)] = 90. Finally, the Modify operation does not change the pixels in the DR, so $Temp_{min}$ and $Temp_{max}$ do not change.

The last two steps of the algorithm convert $BOUND_{min}$ and $BOUND_{max}$ to percentages of imageSize. So, $BOUND_{min}$ is set equal to 60 / 100 = 0.6, and $BOUND_{max}$ is set equal to 90 / 100 = 0.9. The result of the BOUNDS algorithm, then, is that the percentage of pixels that are white in V8 will be in the range [.6, .9]. Since this range does intersect the query range [.5, 1], V8 is returned to the user.

To summarize, the proposed algorithm for processing range queries in a virtual image retrieval system will return three images in response to the query "*Retrieve all images that are at least 50% white*" for the sample database. Image B4 will be returned using conventional histogram techniques. Images V7 and V8 will be returned as a result of the values generated from the BOUNDS algorithm.

CHAPTER 4

TECHNIQUES FOR PROCESSING NEAREST NEIGHBOR QUERIES

This section proposes an algorithm for processing nearest neighbor queries in an image retrieval system that uses virtual images. The specific type of query that the algorithm addresses is "*Retrieve the k images that are nearest to image Q*", where k is a whole number, and Q is the query image. The goal of this approach is to process the nearest neighbor queries utilizing the rules presented in the previous chapter. By meeting this goal, this approach avoids having to instantiate the virtual images since instantiation takes a long time.

4.1. Algorithm for Processing Nearest Neighbor Queries

As described in Chapter 2, conventional systems typically extract and store features of the images in the database as they are inserted. The distance between one of these stored images and the query image Q is based on the comparison of the features extracted from them. This implies that one of the first steps of the retrieval algorithm is to identify and extract the features of Q. This is also the first step in the proposed retrieval algorithm for virtual images.

The proposed approach is similar to the algorithm for processing range queries in that it consists of two other main tasks. The first is to use conventional methods to identify the k binary images that are the closest to the query image based on comparing their respective histograms. The second step is to examine the description of each virtual image and infer if any of them are closer to the query image than the binary images. Again, this means that the system will have to infer the values in the histogram bins for a virtual image, which it does by repeatedly performing range queries on the bins. Consequently, the proposed method of processing nearest neighbor queries utilizes the rules presented for the range queries in the last chapter in order to determine the distance between a virtual image and the query image.

As both the conventional and proposed algorithms for finding the nearest k images to a query image proceed, they need to store the k closest known images at each step. This is accomplished through the maintenance of an array called NEAREST in which the i^{th} element of the array contains two values, the identifier of the i^{th} closest image to Q and its distance to Q. These values are referred to as NEAREST[i].image and NEAREST[i].distance, respectively. So, to determine if an image is one of the k closest neighbors to Q, it can be checked to see if it is smaller than NEAREST[k].distance. This value, then, is always stored in another variable called *smallest.*

The algorithm for processing nearest neighbor queries is shown in Figure 4-1. The first two steps extract the color histogram from the query image, and the next two steps initialize the NEAREST array and the variable smallest to infinity. The next step processes each binary image in the database by comparing their histograms to the one extracted from Q. So, at the completion of Step 4, the NEAREST array will contain the k closest binary images to Q.

The key step in the algorithm is the next step, Step 6, which identifies the virtual images whose distance from the query image is less than the smallest known distances. This step executes the procedure called VIRTUAL_NN, which updates the NEAREST array based on the computed distances of the virtual images. So, when the procedure finishes, the NEAREST array will contain the k closest images to the query image irrespective of how they are stored, and that allows the subsequent step to simply display the images in NEAREST. The VIRTUAL_NN procedure is described in the next section.

```
/* Binary images are compared using histograms, so extract the histogram from the
query image. */
1. Identify parameters (k, Q) from given query
2. Extract the color histogram from image Q and represent it by variable HQ

/* As the algorithm proceeds, the system must track the known k nearest neighbors at all
times. This information is kept in the NEAREST array. The ith element of the array
contains the identifier of the ith known closest image and its distance to Q. The following
code initializes the array. */
3. For i = 1 to k
    3.1. Set the image field of the ith element in NEAREST to null
    3.2. Set the distance field of the ith element in NEAREST to infinity
4. Set smallest equal to the distance field of the kth element in NEAREST

/* Identify k closest binary images using their histograms stored in the database. */
5. For each histogram tuple, H, stored in the database
    5.1. Compute d, the distance between H and HQ.
    5.2. if d < smallest
        5.2.1. Obtain the object id associated with H and call it image
        5.2.2. Insert d and image into the NEAREST array so that the array remains
               sorted based on the distance fields.
        5.2.3. Set smallest equal to the distance field of the kth element in NEAREST

/* Identify virtual images that may be among the k closest */
6. Call VIRTUAL_NN to find the virtual images that should be in NEAREST

/* Retrieve the identified images to the user. Any virtual images must first be
instantiated. */
7. Display the images contained in the first k image fields of the NEAREST array.
```

Figure 4-1. Proposed Algorithm for Processing Nearest Neighbor Queries

4.2. Algorithm for Determining the Distances from Q to the Virtual Images

This section describes the algorithm for computing the distances the query image Q and all of the virtual image in the database. The goal of the algorithm is to compute these distances without having to instantiate any of the virtual images. To accomplish this goal, the algorithm utilizes the rules presented in the last chapter by executing multiple range queries. These range queries provide values that are used to compute the distances from Q to the set of virtual images.

The idea for the proposed nearest neighbor query processing algorithm is based upon submitting a query to the image retrieval system that requests the images that are the most similar to an all black query image, QB. If an image I in the database is 70% black, then the Histogram Intersection of the query image and I is HI(QB, I) = .7. So, the number of black pixels in an image directly relates to the value of the Histogram Intersection between that image and QB. Another way, then, of satisfying the given nearest neighbor query is to find images that are mostly black, which is very similar to the type of range queries processed in the preceding chapter.

The above example indicates that it may be possible to process some nearest neighbor queries by processing various range queries. So, the technique used by the proposed algorithm displayed in Figure 4-2 is to identify and execute such range queries in response to a submitted nearest neighbor query. The algorithm works by repeatedly identifying the bins with the largest values in the histogram of the query image and then performing range queries on those bins.

```
VIRTUAL_NN
/* Initialize the variables used to find the distance between the Query image Q and the virtual images
pctRemaining represents the percentage of the histogram of Q that has not been tested
        Remaining contains the set of virtual images that may be among the k-nearest neighbors
        TotalSum[V] represents the computed Histogram Intersection of V and Q
        numberBins represents the total number of colors within Q
        queryBins represents the bin numbers of the colors contained within Q
        index represents the bin number of the current color being processed*/
1. pctRemaining = 1.00
2. Remaining = Set of Virtual Images in database
3. For each virtual image V in Remaining
   3.1. TotalSum[V] = 0
4. Let queryBins represent the nonzero bins of HQ in sorted order
5. Let numberBins represent the number of nonzero bins in HQ
6. index = 0

7. While (index < numberBins) AND (Remaining ≠ ∅)
   /* Identify the parameters of the range query using the next color in queryBins, called
   currentBin. The query should return those virtual images whose percentage of pixels are close to
   the value in currentBin. */
   7.1. Let currentBin = the next bin in queryBins
   7.2. Let match = the value of the currentBin bin in the histogram of the query image
   7.3. Reduce pctRemaining by match since we are using that portion of the histogram
   7.4. $PCT_{min}$ = MAX(0, match – smallest), which is the minimum percentage of pixels in
        currentBin that a virtual image can have and be a k-nearest neighbor.
   7.5. $PCT_{max}$ = MIN(1, match + smallest), which is the maximum percentage of pixels in
        currentBin that a virtual image can have and be a k-nearest neighbor.

   /* Apply the rules to each virtual image that has not yet been eliminated. */
   7.6. For each virtual image V left in Remaining
        7.6.1.  Apply the BOUNDS algorithm on V for the above range query parameters.

        /* If the virtual image can satisfy the above query, its known distance to the query image
        should be modified with this new information. Those images that cannot satisfy the above
        range query cannot be closer than the known nearest neighbors. */
        7.6.2.  if V satisfies the above range query
                7.6.2.1. increase TotalSum[V] by MIN (match, F($BOUND_{min}$, $BOUND_{max}$))
        7.6.3.  else
                7.6.3.1. Remove V from Remaining

        /* Eliminate the virtual image if there is not enough percentages left to match. */
        7.6.4.  if TotalSum[V] + pctRemaining < 1–smallest
                7.6.4.1. Remove V from Remaining

/* Use the values computed in the TotalSum array as the Histogram Intersection between the query
image and virtual images not eliminated. Compute the distances using these Histogram Intersection
values and update the NEAREST array. */
8. For each virtual image V in Remaining
   8.1. Use TotalSum[V] as the histogram intersection of Q and V
   8.2. Compute the distance from Q to V as 1 – the histogram intersection of Q and V
   8.3. If the distance from Q to V < smallest
        8.3.1.  insert (distance, V) into NEAREST so that the array remains sorted based on the
                distance fields.
        8.3.2.  Set smallest equal to the distance field of the kth element in NEAREST
```

Figure 4-2. VIRTUAL_NN Algorithm for Query Processing of Virtual Images

4.2.1. Algorithm Steps

Steps 1-6 initialize the variables used in the algorithm. The variables pctRemaining, Remaining, TotalSum[V], queryBins, numberBins, and index respectively represent the unused percentage of the query histogram, the set of virtual image IDs not yet eliminated, the histogram intersection of the virtual image V and the query image, the IDs of the nonzero bins of the query histogram in sorted order, the number of IDs in queryBins, and the current element of queryBins.

Step 7 is the main loop of the algorithm. It repeatedly generates range queries based on the values in queryBins. Each iteration of the main loop represents one range query performed by the algorithm. When a set of bounds are computed for a virtual image V using the rules presented in the previous chapter, the minimum bound is added to TotalSum[V]. For example, if the BOUNDS algorithm in the last chapter computes that the value of the current histogram bin for a virtual image will be in the range [.2, .5], 0.2 is added to TotalSum[V]. Again, this sum represents the histogram intersection between Q and V. The virtual images that cannot be one of the k nearest neighbors to Q will be eliminated from consideration by removing their IDs from the variable Remaining. The values in the TotalSum array are used along with the pctRemaining variable to determine which virtual images cannot be one of the nearest neighbors to Q.

Steps 7.1 and 7.2 identify the value of the query histogram at the current bin number in queryBins. Step 7.3 indicates that part of the query histogram is being used by reducing pctRemaining, which represents the unused portion of that histogram. Steps 7.4 and 7.5 compute the boundaries of the range query for this iteration. If a virtual image matched the query image exactly, then the percentage of pixels in currentBin would be equal to the variable *match*. So, the generated range query should look for images that have that *match* pixels in currentBin. Now, consider if the *match* is 60%, and the kth smallest known distance is 20%. If the percentage of pixels of the virtual image in currentBin is known to be less than 40% or more than 80%, then the image is more than 20% different from the query image. This means that the virtual image cannot be one of the k nearest neighbors. Consequently, the algorithm should search for virtual images that have a value in currentBin between 40% and 80% in that situation. Thus, PCT_{min} and PCT_{max} should be defined to be match – smallest and match + smallest, respectively.

Step 7.6 contains a loop that applies the rules for the range query for each virtual image in remaining. Step 7.6.1 applies the query range with the parameters [PCT_{min}, PCT_{max}] as computed earlier. The variable currentBin represents the quantized query color.

Steps 7.6.2 and 7.6.3 perform actions based on whether the current virtual image V satisfies the query. As indicated before, if V cannot satisfy the query, then it cannot be one of the k nearest neighbors of Q. Consequently, it is removed from the list of virtual images in *Remaining*. If V can satisfy the query, then the algorithm increases the value in TotalSum[V]. The reason is that the computation of the Histogram Intersection between two images is accomplished by adding the sums of the minimum values in each of their respective bins. Since the algorithm computed that there is some value, *x*, in bin currentBin for V that is between $BOUND_{min}$ and $BOUND_{max}$, TotalSum[V] should be increased by the minimum of *x* and *match*, which is the percentage of pixels in bin currentBin for Q.

Since the BOUNDS algorithm only produces bounds on the value x, the algorithm in Figure 4-2 needs a method of producing an exact value for x. If the range computed by the BOUNDS algorithm is large, then it is unknown whether MIN(x, *match*) reflects the actual value added for bin currentBin when computing the Histogram Intersection of Q and V. Alternatively, if the computed range is small, then MIN(x, *match*) should be very close to the actual value added for bin currentBin. So, to produce an exact value for x, the algorithm in Figure 4-2 uses a function on the computed bounds that increases as the difference between $\text{BOUND}_{\text{max}}$ and $\text{BOUND}_{\text{min}}$ decreases. Specifically, it computes x as $\text{BOUND}_{\text{max}} \times (1 - (\text{BOUND}_{\text{max}} - \text{BOUND}_{\text{min}}))$.

Step 7.6.4 checks the total in TotalSum for the virtual image. Since the algorithm only adds the minimum bounds computed by the range query rules, the Histogram Intersection total may be so small that the image will not be closer than *smallest*. In this case, the virtual image should be removed from *Remaining*.

Step 8 is the final step of the algorithm. The algorithm updates the NEAREST array according to the values for in TotalSum for each virtual image left in Remaining. So, when this step completes, NEAREST will contain the k closest images to Q.

4.3. Nearest Neighbor Query Processing Example

This section illustrates the use of the algorithm in Figure 4-1 by processing an example query "*Retrieve the 3 images that are the most similar to B1*". The query will be processed using the database of 4 binary and 4 virtual images presented in the previous chapter for range queries. Table 4-1 presents the histograms of the binary images along with their distances to Q using the formula DIST(Q, X) = 1 – HI(Q, X).

Histogram ID	Image ID	Bin_0	Bin_1	Bin_2	Bin_3	Bin_4	Bin_5	Bin_6	Bin_7	Distance to B1
H1	B1	0	0	0	0	0	0	0.6	0.4	0
H2	B2	0.5	0.5	0	0	0	0	0	0	1
H3	B3	0.8	0	0	0	0	0	0	0.2	.8
H4	B4	0	0	0	0	0	0	0.4	0.6	.2

Table 4-1. Histograms of the Binary Images in the Example Database

The first four steps of the algorithm compute the parameters of the nearest neighbor query and initialize the NEAREST array and smallest to infinity. Using the example query defined earlier, k equals 3, and Q equals B1. The NEAREST array would appear as in Row 1 of Table 4-2, where each distance field is infinity and each image field is NULL.

Row	NEAREST 1 distance	NEAREST 1. image	NEAREST 2. distance	NEAREST 2. image	NEAREST 3. distance	NEAREST 3. image
1	∞	∅	∞	∅	∞	∅
2	0	B1	∞	∅	∞	∅
3	0	B1	1	B2	∞	∅
4	0	B1	.8	B3	1	B2
5	0	B1	.2	B4	.8	B3
6	0	B1	.2	B4	.2	V7

Table 4-2. States of NEAREST Array as Nearest Neighbor Algorithm Proceeds

Step 5 of the algorithm in Figure 4-1 contains a loop that processes each of the binary images stored in the database. It uses conventional techniques to identify the k closest binary images meaning that it compares the previously extracted histograms of the images to the histogram of Q.

Table 4-1 indicates that the distance from the query image to B1 is 0, which is because the query image is B1. So, the variable d is set to 0 in Step 4.1, and is compared to the variable smallest in Step 4.2. Since 0 is less than smallest, the value 0 and image B1 are added to the NEAREST array in sorted order based on distance. So, after processing the histogram for B1, the NEAREST array would appear as in Row 2 of Table 4-2. Note that smallest is set equal to NEAREST[3].distance, so it still equals infinity.

The next pass through the loop in Step 5 processes image B2. The distance from Q to B2 is 1, and this is less than smallest. So, the value 1 and image B2 are added to the NEAREST array. The result is after processing the histogram for B2, the NEAREST array would appear as in Row 3 of Table 4-2.

The third pass through the loop in Step 5 processes B3, which is a distance of 0.8 from Q. Again, this is less than smallest, so 0.8 and B3 are added to NEAREST. The result is that the NEAREST array would appear as in Row 4 of Table 4-2. Note that when smallest is set equal to NEARET[3].distance in Step 4.2.3, it becomes equal to 1.

The algorithm now processes the last binary image, B4. The distance from it to Q is 0.2, and this is less than smallest. So, as with the previous images, the value 0.2 and B4 are added to the NEAREST array in sorted order. Since the NEAREST array only needs to track the k closest images, this removes B2 from it leaving the array to appear as it does in Row 5 of Table 4-2.

So, at the conclusion of Step 5, the NEAREST array contains the 3 closest images to Q, B1, B4, and B3. The variable smallest is set to the third closest distance, which is 0.8. So, Step 6 will execute the VIRTUAL_NN algorithm to identify the virtual images whose distances to Q are smaller than 0.8. That algorithm begins by initializing pctRemaining to 1, Remaining to {V5, V6, V7, V8}, and TotalSum[V_i] to 0 for i = 5 to 8.

For Step 4, the histogram of Q is <0, 0, 0, 0, 0, 0, 0.6, 0.4>. The nonzero bins of Q, then, are bins 6 and 7. The variable queryBins contains the ordered list of nonzero bins in Q, so queryBins is {6, 7}. The variable numberBins represents the number of bins in queryBins, so numberBins is set to 2 in Step 5. Finally, index is initialized to 0 in Step 6.

The main portion of the VIRTUAL_NN algorithm is the loop in Step 7. Since index is less than numberBins, and Remaining is not empty, the loop executes. Step 7.1 sets currentBin to queryBins[0], which is 6, and Step 7.2, sets the variable match to the current bin in the query histogram, which is 0.6. Next, Step 7.3 subtracts that total from pctRemaining, causing pctRemaining to now equal 1 – 0.6 = 0.4. This initial processing allows the algorithm to compute PCT_{min} and PCT_{max}. The former is equal to the maximum of 0 and match – smallest. The latter is equal to the minimum of 1 and match + smallest. Since the variable smallest is comparatively large for a percentage, PCT_{min} and PCT_{max} are 0 and 1, respectively.

Step 7.6 performs the range query for the range [0, 1] and bin number 6. After applying the rules for V5, the resulting bounds are [0, 0]. Since this range is in the query range [0, 1], V5 satisfies the query, Consequently, TotalSum[V5] is increased by MIN(0, 0) = 0 in Step 7.6.2.1. Step 7.6.4 tests the value of TotalSum[V5] + pctRemaining, which is 0 + 0.4, and compares it to 1 – smallest, which is 0.2. Since the former value is greater, V5 is not removed from the Remaining set.

After applying the rules for V6, the resulting bounds are [0, 0] as they are for V5. The rest of the loop of Step 7 has the same values, then, which means that V6 is also kept in the Remaining set. When applying the rules for V7, however, the resulting bounds are [.4, .4]. Now, TotalSum[V7] is increased by MIN(0.4, 0.6), which equals 0.4. Since TotalSum[V7] + pctRemaining equals 1, it is greater than smallest, which means that V7 is also kept in Remaining.

Finally, when applying the rules for V8, the resulting bounds are [.1, .4]. TotalSum[V8] is increased by MIN(0.1, 0.6), which is 0.1. TotalSum[V8] + pctRemaining is 0.5, which is greater than 1 – smallest. So, V8 is kept in Remaining.

After the first iteration of the loop in Step 7, TotalSum only changed for the virtual images V7 and V8. So, the algorithm has currently computed the histogram intersections for V7 and V8 to be 0.4 and 0.1, respectively. The histogram intersections for V5 and V6 are still 0.

At this point, the algorithm processes the loop of Step 7 for the next bin listed in queryBins. So, for the second and final iteration of the loop, currentBin becomes bin 7. The value of bin 7 in the query histogram is 0.4, and match is set to this value in Step 7.2. Since all of the query histogram is now being used, pctRemaining reduces to 0 in Step 7.3. Finally, Steps 7.4 and 7.5 set the range query parameters to [0, 1]. Again, the parameters are high because the percentage in the variable smallest is 0.8.

Now the algorithm processes range queries for bin 7 using the query range [0, 1]. The rules for V5 generate a bound of [0, 0], which means that TotalSum[V5] is again increased by 0. Now, when Step 7.4 tests the value of TotalSum[V5] + pctRemaining, it is 0 which means that it is less than 1 – smallest. Consequently, V5 is removed from Remaining in Step 7.6.4.1.

The range query rules for V6 generate a bound of [0.2, 0.2]. Now, TotalSum[V6] is increased by 0.2, which sets it at 0.2. When Step 7.4 tests the value of TotalSum[V6] + pctRemaining, it gets 0.2 + 0 = 0.2. Since this value is not greater than 1 – smallest, V6 is also removed from Remaining.

The rules for V7 generate a bound of [0.6, 0.6]. TotalSum[V7] is increased by MIN(0.6, 0.4) = 0.4. So, TotalSum[V7] is now 0.8. Step 7.6.4 compares the sum of this value and pctRemaining to 1 – smallest. Since the former is greater, V7 is kept in Remaining.

The rules for V8 generate a bound of [0,6, 0.9]. This means that TotalSum[V8] is increased by MIN(0.6, 0.4) = 0.4, which brings its total to 0.5. Like V7, this value added to pctRemaining is greater than 1 – smallest, so V8 is kept in the Remaining set.

So, when Step 7 terminates, there are only two images left in the set Remaining, V7 and V8. The algorithm now moves to Step 8, whose purpose is to adjust the NEAREST array with the known distances. The values in TotalSum are used as the Histogram Intersection between the virtual image and the query image, so HI(V7, Q) is 0.8, and HI(V8, Q) is 0.5. This means that Dist(V7, Q) is 0.2 and Dist(V8, Q) is 0.5. So, after inserting these values in the NEAREST array, the result would appear as in Row 6 of Table 4-2. This means that the proposed algorithm returns B1, B4, and V7 as the 3 closest neighbors to Q.

4.4. Discussion of Example

Note that from the example in the previous section, the final values of TotalSum for V5, V6, V7, and V8 were 0, 0.2, 0.8, and 0.5, respectively. These values represent the computed Histogram Intersection between the images and the query image, B1. The purpose of this section is to present the reasons for these totals using the descriptions of the example virtual images presented in Chapter 3.

From the histogram of the query image, 60% of its pixels are mapped to bin 6, and 40% of its pixels are mapped to bin 7. Recall from the previous chapter that the color (255, 255, 255) also mapped to bin 7 in the example virtual image retrieval system. So, the algorithm should return those virtual images whose descriptions indicate that they have pixels whose intensities are mapped mostly to bins 6 and 7.

Virtual image V5 is created by cropping 50 pixels from its base image B2. According to the histogram displayed in Table 4-1, this image does not have any pixels in bins 6 or 7, so V5 does not have any pixels in those bins, either. As a result, the histogram intersection between V5 and B1 should be 0, which is the value generated by the proposed algorithm.

According to the description of virtual image V6, it is created as a transformation of its base image B3. This image has 20% of its pixels in bin 7, but no pixels in bin 6. The description of V6 indicates that it may change up to 20% of the pixels in B3 to the color (255, 255, 255), so it may have even more pixels in bin 7. Consequently, the proposed algorithm computes the histogram intersection of Q and V6 to be higher than the histogram intersection of Q and V5, which implies that V6 is more similar to Q than V5.

Virtual images V7 and V8 are created as transformations of base image B4, which is only a distance of 0.2 from the query image. By reviewing the histograms in Table 4-1, it can be seen that for image B4 to match image B1, it needs to increase the number of pixels in bin 6 and decrease the number of pixels in bin 7. Image V8 is created by changing a portion of B4 to the color (255, 255, 255), which maps to bin 7. As a result, it can only increase the number of pixels in bin 7, which means that it cannot be any closer to B1 than B4.

Alternatively, the description of V7 implies that it increases a portion of B4. Since the proposed algorithm cannot determine the effects of this operation, it assumes that the resulting image will have the same histogram as the base image. The result is that V7 is computed to be only a distance of 0.2 from the query image. So, V7 is assumed to be the closest image to Q out of all of the virtual images.

CHAPTER 5

DATA STRUCTURE FOR SPEEDING UP RETRIEVAL PROCESSING

As described in Chapter 2, systems that use conventional approaches such as histograms to retrieve images are able to process submitted retrieval queries without having to access each image in the underlying database. This is frequently accomplished by using an index such as an R-tree [Gutt, 1984] or other type of access method that clusters the data elements into sections of the multidimensional data space of the histograms. Searching is then performed by accessing nodes in the data structure that represent those sections. By quickly identifying sections of the multidimensional space that cannot contain any histograms of images that satisfy the given query, the query processing algorithm can avoid accessing the data elements contained in those sections.

In contrast to the histogram-based approaches, the rule-based algorithms for processing retrieval queries contain steps for determining if each virtual image satisfies the given query and should therefore be retrieved from the database. The steps involve accessing a set of rules for each image editing operation stored in the description of a virtual image in order to determine bounds on the percentage of pixels whose intensities quantize to a given bin. So, the proposed algorithms must access every virtual image in a database as well as every editing operation within a virtual image description to process the given queries. As an alternative, the following sections propose an approach for processing retrieval queries that does not have to access every editing operation contained in the database yet still produces the same results as the algorithms proposed in the previous chapters.

5.1. Properties of Rules for Editing Operations

To present the proposed technique for reducing the number of editing operations that have to be accessed in order to process range queries, it is first necessary to consider the characteristics of the rules that are applied for each operation. Each rule produces new maximum and minimum bounds on the percentage of pixels that may be in a given histogram bin for a virtual image. The virtual image is retrieved if these computed bounds intersect the desired range specified by the query. The algorithm produces these bounds by computing three values, the maximum number of pixels that are in the histogram bin, the minimum number of pixels that are in the histogram bin, and the total number of pixels in the virtual image. So, some characteristics of the proposed rules can be determined by finding characteristics in these three values.

Several of the proposed rules only increase the maximum bound and decrease the minimum bound on the percentage of pixels contained in the desired histogram bin for a virtual image. These rules accomplish this by only increasing the maximum bound, $\text{BOUND}_{\text{max}}$, and decreasing the minimum bound, $\text{BOUND}_{\text{min}}$, on the number of pixels in the bin, while keeping the total number of pixels, imageSize, in the virtual image constant. The result is that these rules will only widen the range specified by the minimum bound and maximum bounds. Rules that exhibit this characteristic are called *bound-widening* rules, and they are presented in the following section.

5.1.1. Bound-Widening Rules

In the proposed range query processing algorithm, there are rules presented for the four editing operations Combine, Modify, Mutate, and Merge. The following theorems will review the rules for each of the operations and determine if they are bound-widening. In each section, the variables $\text{before}_{\text{min}}$ and $\text{before}_{\text{max}}$ will respectively represent the values of the percentages $\text{BOUND}_{\text{min}}$/imageSize and $\text{BOUND}_{\text{max}}$/imageSize before applying the rule for the particular editing operation. Similarly, the variables $\text{after}_{\text{min}}$ and $\text{after}_{\text{max}}$ will respectively represent the values of the percentages after applying the rule. Therefore, to demonstrate that a rule is bound-widening, it is necessary to prove that $\text{before}_{\text{min}} \geq \text{after}_{\text{min}}$, and $\text{before}_{\text{max}} \leq \text{after}_{\text{max}}$.

Theorem 5.1. *The rule for the Combine operation is a bound-widening rule.*

Proof: Let the values of $\text{before}_{\text{min}}$ and $\text{before}_{\text{max}}$ equal $\text{BOUND}_{\text{min}}$/imageSize and $\text{BOUND}_{\text{max}}$/imageSize, respectively. The only rule for the Combine operation does not change the values for the $\text{BOUND}_{\text{min}}$, $\text{BOUND}_{\text{max}}$, and imageSize variables. So, after applying the rule, the values of $\text{after}_{\text{min}}$ and $\text{after}_{\text{max}}$ will also equal $\text{BOUND}_{\text{min}}$/imageSize and $\text{BOUND}_{\text{max}}$/imageSize. Thus, $\text{before}_{\text{min}} \geq \text{after}_{\text{min}}$, and $\text{before}_{\text{max}} \leq \text{after}_{\text{max}}$.

Theorem 5.2. *The rules for the Modify operation are bound-widening rules.*

Proof: There are several rules proposed for the Modify operation depending on the values in its parameters. None of the rules changes the value of imageSize. So, it is possible to define the value of $\text{after}_{\text{min}}$ to be new_{min}/imageSize where new_{min} represents the new value of the variable $\text{BOUND}_{\text{min}}$ after the application of the rules for Modify. Since $\text{before}_{\text{min}}$ equals $\text{BOUND}_{\text{min}}$/imageSize, $\text{before}_{\text{min}} \geq \text{after}_{\text{min}}$ can be proven by showing $\text{BOUND}_{\text{min}} \geq \text{new}_{\text{min}}$.

Only one of the proposed rules changes the value of $\text{BOUND}_{\text{min}}$, and it sets it equal to MAX[temp_{min}, MAX(0, $\text{BOUND}_{\text{min}}$–DR_Size)]. This expression has three terms, temp_{min}, 0, and $\text{BOUND}_{\text{min}}$–DR_Size. So, after applying the rule for the Modify operation, new_{min} may have one of four possible values, $\text{BOUND}_{\text{min}}$, temp_{min}, 0, or $\text{BOUND}_{\text{min}}$–DR_Size. $\text{BOUND}_{\text{min}}$ represents the number of pixels with intensities that quantize to bin HB, so it will always be greater than or equal to 0. In addition, DR_Size represents the number of pixels in the Defined Rectangle, so it can never be negative. This means that $\text{BOUND}_{\text{min}}$ is always greater than or equal to $\text{BOUND}_{\text{min}}$–DR_Size. Thus, it can quickly be shown that new_{min} will be less than or equal to $\text{BOUND}_{\text{min}}$ if new_{min} equals $\text{BOUND}_{\text{min}}$, 0, or $\text{BOUND}_{\text{min}}$–DR_Size.

Now consider if new_{min} equals temp_{min}, which represents a limit on how far the value of $\text{BOUND}_{\text{min}}$ may decrease. The BOUNDS algorithm always maintains that temp_{min} is less than or equal to $\text{BOUND}_{\text{min}}$ by computing it as $\text{BOUND}_{\text{min}}$–DR_Size every time $\text{BOUND}_{\text{min}}$ could be set to a value lower than it. So, if new_{min} is equal to temp_{min}, then new_{min} will be less than or equal to $\text{BOUND}_{\text{min}}$. The result is that every possible value for new_{min} is less than or equal to $\text{BOUND}_{\text{min}}$, which means $\text{before}_{\text{min}} \geq \text{after}_{\text{min}}$.

Let new_{max} equal the new value of the variable $\text{BOUND}_{\text{max}}$ after the application of the rules for Modify. As with the minimum bounds, $\text{before}_{\text{max}} \leq \text{after}_{\text{max}}$ can be proven by showing $\text{BOUND}_{\text{max}} \leq \text{new}_{\text{max}}$. The rules for the Modify operation change the $\text{BOUND}_{\text{max}}$ variable to the expression MIN[temp_{max}, MIN(imageSize, $\text{BOUND}_{\text{max}}$+DR_Size)], which means that the only possible values for new_{max} are $\text{BOUND}_{\text{max}}$, $\text{BOUND}_{\text{max}}$+DR_Size, imageSize, and temp_{max}.

If new_{max} is $BOUND_{max}$, then it will be greater than or equal to $BOUND_{max}$. Since, DR_Size is never negative, the same is true if new_{max} equals $BOUND_{max}$+DR_Size. Since $BOUND_{max}$ represents the number of pixels in a virtual image that quantize to bin HB, it can never be greater than the total number of pixels in that virtual image. So, $BOUND_{max}$ will always be less than or equal to imageSize. Finally, as with $temp_{min}$ for $BOUND_{min}$, $temp_{max}$ represents a limit on how much $BOUND_{max}$ may increase, which implies that the BOUNDS algorithm always keeps $temp_{max}$ greater than or equal to $BOUND_{max}$.

Since every possible value for new_{max} is always greater than or equal to $BOUND_{max}$, $before_{max} \leq after_{max}$. Thus, the value of $BOUND_{min}$/imageSize never increases, and the value of $BOUND_{max}$/imageSize never decreases, which means that all of the rules for the Modify operation are bound-widening.

Theorem 5.3. *The rules for the Mutate operation are bound-widening rules.*

Proof: Consider the rule for the Mutate operation when the imageSize variable is equal to the DR_Size. The proposed rule changes the values in the $BOUND_{min}$, $BOUND_{max}$, and imageSize variables by multiplying them by the same constant. So, both of the values $BOUND_{min}$/imageSize and $BOUND_{max}$/imageSize remain the same after the application of the rule. As a result, $before_{min}$ equals $after_{min}$, and $before_{max}$ equals $after_{max}$, which means that the rule for the Mutate operation when imageSize equals DR_Size is bound-widening.

When a rigid body transformation is specified in the parameters of the Mutate operation, the rule keeps the imageSize variable constant. In addition, it changes

$\text{BOUND}_{\text{min}}$ to MAX[temp_{min}, MAX(0, $\text{BOUND}_{\text{min}}$–DR_Size)] and $\text{BOUND}_{\text{max}}$ to MIN[temp_{max}, MIN(imageSize, $\text{BOUND}_{\text{max}}$+DR_Size)], which means that it never increases $\text{BOUND}_{\text{min}}$ and never decreases $\text{BOUND}_{\text{max}}$ as explained in the theorem for the Modify operation. Since imageSize remains constant, the value of $\text{BOUND}_{\text{min}}$/imageSize never increases, and the value of $\text{BOUND}_{\text{max}}$/imageSize never decreases, which means that the rule for a rigid body transformation is bound-widening.

Finally, the remaining rule for the Mutate operation keeps $\text{BOUND}_{\text{min}}$, $\text{BOUND}_{\text{max}}$, and imageSize constant. Therefore, the value of $\text{BOUND}_{\text{min}}$/imageSize never increases, and the value of $\text{BOUND}_{\text{max}}$/imageSize never decreases, which again means that this rule is bound-widening. Thus, all three proposed rules for the Mutate operation are bound-widening.

Theorem 5.4. *The rule for the Merge operation when the target parameter is NULL is bound-widening.*

Proof: When the target parameter of the Merge operation is NULL, the rule changes the value of imageSize, $\text{BOUND}_{\text{max}}$, and $\text{BOUND}_{\text{min}}$. Let the values of $\text{before}_{\text{min}}$ and $\text{before}_{\text{max}}$ equal $\text{BOUND}_{\text{min}}$/imageSize and $\text{BOUND}_{\text{max}}$/imageSize, respectively. The values of imageSize, $\text{BOUND}_{\text{max}}$, and $\text{BOUND}_{\text{min}}$ after the rule for the Merge operation is applied must be determined in order to determine $\text{after}_{\text{min}}$ and $\text{after}_{\text{max}}$.

When the parameters of the Merge operation specify a NULL target image, the variable imageSize changes to DR_Size. The new value of $\text{BOUND}_{\text{min}}$ is MAX[0, DR_Size – (imageSize – $\text{BOUND}_{\text{min}}$)], which contains two possible terms, 0 and DR_Size – (imageSize – $\text{BOUND}_{\text{min}}$). So, the value of $\text{after}_{\text{min}}$ may either be 0/DR_Size, which is 0, or (DR_Size – (imageSize – $\text{BOUND}_{\text{min}}$))/imageSize.

Consider the first case when $after_{min}$ is 0. Since 0 is the smallest possible value for the value in bin HB, $before_{min} \geq 0$. Thus, if $after_{min}$ is 0, $before_{min} \geq after_{min}$.

Alternatively, consider when $after_{min}$ is (DR_Size – (imageSize – $BOUND_{min}$))/imageSize, and compare it to $before_{min}$, $BOUND_{min}$/imageSize. Multiplying both expressions by DR_Size gives $before_{min}$ a value of ($BOUND_{min}$ × DR_Size)/imageSize and $after_{min}$ a value of DR_Size – (imageSize – $BOUND_{min}$). Next, subtracting DR_Size from both expressions gives a total of ($BOUND_{min}$ × DR_Size)/imageSize – DR_Size for $before_{min}$ and a total of $BOUND_{min}$ – imageSize for $after_{min}$.

The variable $before_{min}$ is now equivalent to ($BOUND_{min}$ × DR_Size)/imageSize – (imageSize × DR_Size)/imageSize, which equals (($BOUND_{min}$ × DR_Size) – (imageSize × DR_Size))/imageSize. By factoring DR_Size in the numerator, $before_{min}$ can be expressed as DR_Size/imageSize × ($BOUND_{min}$ – imageSize).

The only difference between the two variables is that $before_{min}$ is multiplied by DR_Size/imageSize. A Defined Rectangle is always created by cropping a portion of the image, so the number of pixels in the Defined Rectangle will always be less than or equal to the number of pixels in the image, which means DR_Size ≤ imageSize. So, (DR_Size/imageSize) ≤ 1. Similarly, $BOUND_{min}$ will always be less than or equal to the number of pixels in the image, so ($BOUND_{min}$ – imageSize) ≤ 0. The result is that multiplying ($BOUND_{min}$ – imageSize) by (DR_Size/imageSize) gives a value that is greater than or equal to ($BOUND_{min}$ – imageSize). Thus, $before_{min} \geq after_{min}$.

The above paragraphs indicate that $before_{min} \geq after_{min}$ irrespective of whether $BOUND_{min}$ becomes 0 or DR_Size – (imageSize – $BOUND_{min}$). Now, consider the maximum bound specified by the rule, which sets $\text{BOUND}_{\max}$ to the minimum of $\text{BOUND}_{\max}$ and DR_Size. The result is that $\text{after}_{\max}$ will equal either $\text{BOUND}_{\max}$/DR_Size or DR_Size/DR_Size.

For the first case, since DR_Size $\leq$ imageSize, $\text{BOUND}_{\max}$/DR_Size $\geq$ $\text{BOUND}_{\max}$/imageSize. So, if $\text{after}_{\max}$ equals $\text{BOUND}_{\max}$/DR_Size, $\text{before}_{\max} \leq \text{after}_{\max}$. Alternatively, if the second case is true, then $\text{after}_{\max}$ equals DR_Size/DR_Size, which means it equals 1. Since this value represents a bound on the percentage of pixels whose intensities quantize to bin HB, its maximum possible value is 1. Thus, $\text{before}_{\max}$ must be less than or equal to 1, which means $\text{before}_{\max} \leq \text{after}_{\max}$.

So, irrespective of whether $\text{after}_{\max}$ equals $\text{BOUND}_{\max}$/DR_Size or DR_Size/DR_Size, $\text{before}_{\max} \leq \text{after}_{\max}$. In addition, $\text{before}_{\min} \geq \text{after}_{\min}$ as described earlier, so the rule for the Merge operation when the target parameter is NULL is bound-widening.

Theorem 5.5. *The rule for the Merge operation when the target parameter is not NULL is not bound-widening.*

Proof: When the target parameter of the Merge operation is not NULL, the associated rule may update the value of imageSize, $\text{BOUND}_{\max}$, and $\text{BOUND}_{\min}$. Let the values of $\text{before}_{\min}$ and $\text{before}_{\max}$ equal $\text{BOUND}_{\min}$/imageSize and $\text{BOUND}_{\max}$/imageSize, respectively. The values of imageSize, $\text{BOUND}_{\max}$, and $\text{BOUND}_{\min}$ after the rule for the Merge operation is applied must be determined in order to determine $\text{after}_{\min}$ and $\text{after}_{\max}$.

It is possible that this rule for the Merge operation does not change the value of imageSize when the base image and the target image are the same size. The rule for the

BOUND_{max} variable changes its value to the sum of MIN(TargetHB, MAX(0, TargetSize – DR_Size)) and the minimum of BOUND_{max} and DR_Size. The first term could equal 0 if TargetHB is 0. This would mean that the new value of BOUND_{max} would be MIN(BOUND_{max}, DR_Size). So, when DR_Size < BOUND_{max}, the value of after_{max} would be DR_Size/imageSize. Since DR_Size < BOUND_{max}, DR_Size/imageSize < BOUND_{max}/imageSize, which means that $\text{after}_{max} < \text{before}_{max}$. Thus, the rule for the Merge operation when the target parameter is not null is not bound-widening.

5.1.2. Technique for Speeding up Retrieval Query Processing

To illustrate the importance of bound-widening rules, consider applying the proposed range query processing algorithm to a virtual image V where all of the rules for the editing operations in its description are bound-widening. When the BOUNDS algorithm completes, the BOUND_{min} and BOUND_{max} values are divided by imageSize to obtain the minimum and maximum percentages. If the range [BOUND_{min}/imageSize, BOUND_{max}/imageSize] intersects the desired query range [PCT_{min}, PCT_{max}], then the virtual image is returned by the query processing algorithm.

The BOUNDS algorithm begins by initializing both BOUND_{min} and BOUND_{max} to the value in histogram bin HB corresponding to the base image of V and initializing imageSize to be the total number pixels in the base image of B. So, if the percentage of pixels in bin HB in the base image of V is basePercentage, and the total number of pixels in the base is baseSize, then the initial values of BOUND_{min}, BOUND_{max}, and imageSize are basePercentage × baseSize, basePercentage × baseSize, and baseSize, respectively.

This means that the initial maximum and minimum percentages generated by the BOUNDS algorithm are equal to (basePercentage×baseSize)/baseSize = basePercentage.

Now, consider if the base image of V satisfies the given query. This means that basePercentage, which is the value in bin HB, is within the desired query range [PCT_{min}, PCT_{max}]. Let $Final_{min}$ and $Final_{max}$ represent the values of $BOUND_{min}$/imageSize and $BOUND_{max}$/imageSize after the application of the BOUNDS algorithm. Since all of the rules for the editing operations in the description of V are bound-widening rules, the value $Final_{min}$ will be less than or equal to the initial value of $BOUND_{min}$/imageSize, and the value $Final_{max}$ will be greater than or equal to the initial value of $BOUND_{max}$/imageSize. Both initial values were basePercentage, so the range [$Final_{min}$, $Final_{max}$] contains basePercentage. Since this value is also within [PCT_{min}, PCT_{max}], the ranges [$Final_{min}$, $Final_{max}$] and [PCT_{min}, PCT_{max}] must intersect. Thus, V would be retrieved by the proposed range query processing algorithm.

The above information implies that if a virtual image has a base image that satisfies the given query and has only editing operations that correspond to bound-widening rules, then the proposed algorithm will retrieve the virtual image. This determination can be made without the BOUNDS algorithm being executed, meaning that the rules for the editing operations do not have to be applied. Therefore, with the above information, it is possible to develop an algorithm that will produce the same results as the proposed range query processing algorithm without applying the rules to each editing operation in the descriptions of the virtual images in the database.

5.2. Proposed Data Structure

To avoid applying some of the rules in the description of a virtual image V, a data structure is needed that stores whether the base image of V satisfies a given query and whether the editing operations in the description of V have bound-widening rules. Consequently, this section proposes a data structure that will store this information. The proposed data structure consists of two different components called the Main Component and the Unclassified Component.

The Main Component contains a list of the virtual images whose descriptions contain editing operations that have only bound-widening rules proposed for it. These virtual images are clustered together based upon the base images that are listed in the first lines of their respective descriptions, meaning that two virtual images are clustered together if and only if they have the same base image. Each cluster contains the histogram identifier corresponding to its associated base image. Therefore, each element of the Main Component is composed of a tuple <H_id, V_list> where

H_id – Histogram identifier
V_list – List of identifiers of virtual images that are derived from the base image corresponding to H_id

Using the Main Component, the system can identify images that satisfy a given range query by accessing each stored histogram identifier and checking if the associated binary image has a value in the desired bin that is within the query range. If so, the system can immediately return the identifier of the binary image as well as the identifiers in V_list.

Some of the virtual images may have descriptions that contain at least one editing operation whose corresponding rule is not bound-widening. The identifiers of such virtual images are stored in the Unclassified Component. To process these identifiers, the system must apply each of the rules to determine the minimum and maximum bounds on the number of pixels whose intensities quantize to the desired bin of the range query. So, the system will execute the BOUNDS algorithm for each virtual image identifier in the Unclassified Component.

5.2.1. Creation of Proposed Data Structure

The proposed data structure can be constructed as images are inserted into the database. Each time a binary image is inserted, the identifier for its corresponding histogram should be added to the Main Component. The list of histogram identifiers should be kept sorted to make it easier to search for a specific histogram.

```
/* Identify the histogram of the base image of the input virtual image V */
1. Identify the base image using the first line of the syntax of virtual image V
2. Access the histogram corresponding to the base image

/* Analyze all of the operations in V to determine if they are all bound-widening. */
3. While ((V has more ops) and (hist ≠ UNCLASSIFIED))
    3.1. Access the rule for the next operation in V
    3.2. If the rule is not bound-widening
        3.2.1. Mark the virtual image V as unclassified

/* If all operations in V are bound-widening, add V to the Main Component, otherwise,
add it to Unclassified. */
4. If V has been marked as unclassified
    4.1. Append the identifier of V to the Unclassified Component
5. else
    5.1. Find location in Main Component referring to the base image
    5.2. Append the identifier of V to the list of virtual images at the above location
```

Figure 5-1. Insertion Algorithm for Proposed Data Structure

Each time a virtual image is inserted into the database, the system needs to determine whether it should be added to the Main Component or the Unclassified

Component. To make this determination, the description of the virtual image must be analyzed in order to identify if it contains any rules that are not bound-widening. If so, then the identifier of the virtual image is added to the Unclassified Component. If all of the rules are bound-widening, then the identifier is added to the cluster in the Main Component corresponding to the base image contained in its description. An algorithm for performing this insertion is displayed in Figure 5-1.

Figure 5-2 displays the states of the proposed data structure as each element of the example database is inserted into the system. Figures 5-2a-d display the data structure as the binary images B1, B2, B3, and B4 are inserted into the underlying database, and Figures 5-2e-h display the data structure as the virtual images V5, V6, V7, and V8 are inserted. So, after binary image B1 is inserted into the database in Figure 5-2a, its histogram ID is added to the Main Component with a NULL V_list. After binary image B2 is inserted in Figure 5-2b, the Main Component has two histogram identifiers H1 and H2 with both of their corresponding V_list entries set to NULL. This pattern continues through Figure 5-2d, which displays the Main Component having the four histogram identifiers corresponding to the four binary images in the database.

Figure 5-2e displays the data structure after V5 is inserted into the database. The only editing operation in the description of V5 is the Merge Operation with a NULL target parameters, so all of its rules are bound-widening. Thus, the image ID of V5 should be added to the Main Component. It is added into the cluster of the V_list corresponding to H2 since H2 is the histogram ID corresponding to the base image of V5. Figure 5-2f displays the data structure after V6 is inserted into the database. Again, all of its editing operations correspond to rules that are bound-widening, so it is added to the

Main Component to the cluster corresponding to its base image. This pattern continues through Figure 5-2h in which virtual image ID V8 is added to the cluster of H4 since the base image of V8 is B4. If any of the virtual images contained an editing operation that was bound-widening, its ID would have been added to the Unclassified Component *instead* of the Main Component.

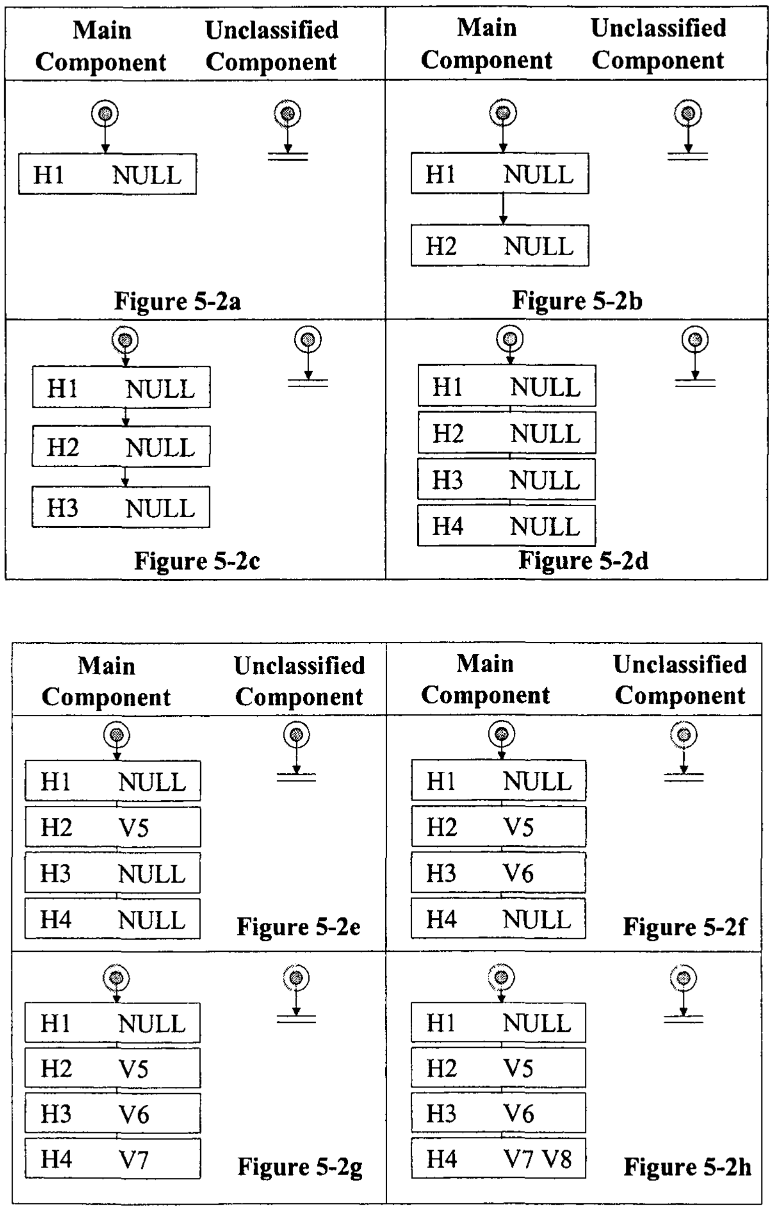


Figure 5-2. Example Data Structure as Images are Inserted into Database

5.3. Range Query Processing Algorithm

The proposed data structure arranges the virtual images based on whether all of their rules are bound-widening. This section presents an algorithm in Figure 5-3 that uses the proposed data structure to avoid applying some of the rules for the editing operations stored within the descriptions of the virtual images in the database. The first three steps of the algorithm are used to identify the desired histogram bin from the query, and they are the same as the steps for the range query processing algorithm proposed in Chapter 3 that does not use the data structure.

The fourth step in the algorithm sequentially accesses each cluster in the Main Component and checks to see if its corresponding histogram satisfies the given query. If so, the binary image identifier in the histogram as well as the virtual image identifiers in the associated list are all added to the satisfying set, which is performed in Steps 4.2.1-4.2.3. Note that the virtual images in the associated list are retrieved without having to execute the BOUNDS algorithm, which means that the rules do not have to be applied for the editing operations in their descriptions. Alternatively, if the histogram does not satisfy the query, then the BOUNDS algorithm must be performed on each element in the associated virtual image list in order to determine if it should be retrieved by the algorithm.

The fifth step sequentially access every virtual image whose identifier is in the Unclassified Component and executes the BOUNDS algorithm in order to determine whether each satisfies the given query. If so, then the identifier is added to the set of retrieved images, results. When this step ends, the results set will contain all of the images that satisfy the given query, so the final step of the algorithm is to display them.

```
/* Initialize the parameters of the given query */
1. Set results = ∅
2. Compute parameters ($PCT_{min}$, $PCT_{max}$, $C_Q$) from query syntax
3. Compute Histogram Bin (HB) from query color $C_Q$ using quantization formula

/* Identify virtual images in Main Component that satisfy the query */
4. For each element <H_id, V_list> in the Main Component

    /* Determine if the binary image of each element satisfies the given query. */
    4.1. pixels = the value in bin HB of histogram H_id

    /* If the binary image does satisfy the query, all elements of V_list satisfy the
    query as well */
    4.2. If ((pixels > $PCT_{min}$) and (pixels < $PCT_{max}$))
        4.2.1. B = image ID corresponding to H_id
        4.2.2. Add B to results
        4.2.3. Add the elements in V_list to results

    /* If the binary image does not satisfy the query, the BOUNDS algorithm
    must be applied to each virtual image referenced in V_list to determine if it
    satisfies the query.. */
    4.3. else
        4.3.1. For each V in V_list
            4.3.1.1. Execute the BOUNDS algorithm for V
            4.3.1.2. If bounds overlap [$PCT_{min}$, $PCT_{max}$]
                4.3.1.2.1. Add V to set results

/* The BOUNDS algorithm must be applied to each virtual image in the
Unclassified Component to determine if it satisfies the query. */
5. For each element V in the Unclassified Component
    5.1. Execute the BOUNDS algorithm for V
    5.2. If bounds overlap [$PCT_{min}$, $PCT_{max}$]
        5.2.1. Add V to results

/* Display the retrieved images by rendering the binary images and instantiating
the virtual images. */
6. Display images in results set
```

Figure 5-3. Range Query Processing Algorithm Using Proposed Data Structure

Histogram ID	Image ID	Bin_0	Bin_1	Bin_2	Bin_3	Bin_4	Bin_5	Bin_6	Bin_7
H1	B1	0	0	0	0	0	0	0.6	0.4
H2	B2	0.5	0.5	0	0	0	0	0	0
H3	B3	0.8	0	0	0	0	0	0	0.2
H4	B4	0	0	0	0	0	0	0.4	0.6

Table 5-1. Histograms of the Binary Images in the Example Database

5.3.1. Range Query Processing Algorithm Steps

The following example illustrates using the proposed data structure to process the example range query "*Retrieve all images that are at least 50% white*". The query is applied to the example database used in Chapter 3. The histograms of the binary images in the database are displayed in Table 5-1.

As indicated earlier, the first three steps of the range query processing algorithm that uses the proposed data structure are the same as the algorithm proposed in Chapter 3. The first step initializes the satisfying set SQ to be empty. The next step identifies the parameters of the example query, which produces PCT_{min} equal to 0.5, PCT_{max} equal to 1.0, and C_Q equal to (255, 255, 255). Finally, the last step quantizes color C_Q to bin number 7.

Step 4 processes the images contained in the Main Component by executing a loop for each existing <H_id, V_list> tuple. The first tuple is <H1, NULL>, which means that H_id equals H1 and V_list is NULL. Step 4.1 accesses bin 7 of histogram H1 and sets pixels equal to its value, which is 0.4. Since this value is not within the query range [0.5, 1], the else condition in Step 4.3 executes. Step 4.3.1 executes a loop for each element in V_list, but since it is NULL, there are no elements. So, the loop does not execute.

The second tuple in the Main Component is <H2, V5>. Since H_id is H2, Step 4.1 sets pixels to the value in bin 7 of histogram H2, which is 0. As before, this value is not within the query range [0.5, 1], so the else condition of Step 4.3 executes. Since set V_list contains V5, Step 4.3.1 executes for it. Step 4.3.1.1 executes the BOUNDS algorithm for V5 yielding a range of [0, 0] as it did in Chapter 3. This range does not overlap the query range, so V5 is not added to set SQ.

The third tuple in the Main Component is <H3, V6>. The variable H_id is now H3, so 4tep 5.1 sets pixels to the value in bin 7 of that histogram, which is 0.2. As in the previous tuples, this value is not in the query range [0.5, 1], so Step 4.3.1 executes for each element in the variable V_list. The variable contains V6, so the BOUNDS algorithm executes for the associated virtual image in Step 4.3.1.1. As indicated in Chapter 3, the result of applying the algorithm is the range [0.2, 0.4]. Since this range does not intersect the query range, V6 is not added to set SQ.

The last tuple in the Main Component is <H4, V7 V8>. The variable H_id is now H4, so Step 4.1 sets pixels to the value in bin 7 of that histogram, which is 0.6. Unlike the previous tuples, this value is within the query range [0.5, 1], so Step 4.2.1 executes. This step obtains the identifier B4 from the current H_id, and then Step 4.2.2 adds it to set SQ. Finally, Step 4.2.3 adds all of the elements in V_list to the satisfying set, so V7 and V8 are added to set SQ.

The fifth step in the algorithm processes the virtual images that are contained in the Unclassified Component. The algorithm executes the BOUNDS algorithm for each virtual image identifier contained within the Unclassified Component. Figure 5-2h displays the contents of the proposed data structure using the example database. In the figure, the Unclassified Component is NULL, which means that it does not contain any virtual image identifiers. Consequently, the loop in Step 4 does not execute.

When Step 5 terminates, set SQ contains B4, V7, and V8, which are the images that satisfy the given query. Again, these are the same results as produced by the algorithm proposed in Chapter 3. This algorithm concludes by displaying the images in set SQ for the user.

To summarize, when using the proposed data structure to process the range query "*Retrieve all images that are at least 50% white*" on the example database, the BOUNDS algorithm only has to be executed for V5 and V6. This is in contrast to the proposed range query processing algorithm in Chapter 3, which must execute the BOUNDS algorithm for all of the virtual images, V5, V6, V7, and V8.

CHAPTER 6

HISTOGRAM-BASED APPROACHES FOR SEARCHING BY COLOR

The approach presented in the previous chapters presents an approach for processing content-based searches using rules based on the description of virtual images. As with the rule-based approach, each of the approaches in this chapter extract and store histograms from the binary images in the database. These approaches, however, differ in how they process images stored virtually.

6.1. Virtual Storage with Instantiation while Searching (VSIS) Approach

The first approach for searching virtual images based on color is called the Virtual Storage with Instantiation while Searching (VSIS) approach, and its strategy is to convert virtual images into a binary format during searching. Once the images are instantiated, the query processor can extract histograms from them as in the conventional approach. The VSIS algorithm for processing range queries is displayed in Figure 6-1, and its first four steps are the same as in the rule-based algorithms for processing range queries since they process binary images in the same manner. Step 5 of the VSIS algorithm for processing range queries is a loop that executes once for each virtual image. In the loop, the virtual image is instantiated, and then its histogram is extracted so that it can be checked to determine if it is within the desired query range. The instantiated versions of those that do satisfy the query are saved so that they can be retrieved in the final step.

```
/* Identify the desired histogram bin from the parameters of the range query */
1. Initialize Results to ∅
2. Compute parameters ($PCT_{min}$, $PCT_{max}$, $C_Q$) from given query
3. Compute Histogram Bin (HB) from query color $C_Q$

/* Search stored histogram bins */
4. For each histogram tuple stored in the database
    4.1 If bin HB is within query range [$PCT_{min}$, $PCT_{max}$]
        4.1.1 Add image ID stored in histogram tuple to Results

/* Determine the virtual images that satisfy the query by instantiating them and
extracting their histograms. Then, process the histograms in the same manner as for the
binary images. */
5. For each Virtual Image
    5.1 Instantiate Virtual Image
    5.2 Extract Histogram from Instantiated Image
    5.3 If bin HB is within query range [$PCT_{min}$, $PCT_{max}$]
        5.3.1 Add image ID stored in histogram tuple to Results
        5.3.2 Save Instantiated Image for Step 6

/* Render all retrieved images since any retrieved virtual image was converted to a
binary format in Step 5. */
6. Display images contained in Results set
```

Figure 6-1. VSIS Algorithm for Processing Range Queries

The VSIS algorithm for processing nearest neighbor queries is displayed in Figure 6-2, and the first five steps are similar to the rule-based approach since it again processes binary images in the same manner. Step 6 of the VSIS algorithm is a loop that executes once for each virtual image in the database. During each iteration, the algorithm instantiates the virtual image, extracts a histogram from it, and computes the distance between that histogram and the one extracted from the query image. The final step of the above algorithm displays the k nearest images. As with the range query processing algorithm, since the virtual images have already been instantiated, the VSIS algorithm can simply load and display each instantiated virtual image that satisfies the query.

```
/* Identify the parameters of the nearest neighbor query. */
1. Identify the parameters k and Q from given query
2. Let HQ be Histogram Extracted from the Query Image Q

/* As the algorithm proceeds, the system must track the known k nearest neighbors at all
times. This information is kept in the NEAREST array. The ith element of the array
contains the identifier of the ith known closest image and its distance to Q. The following
code initializes the array. */
3. For i = 1 to k
    3.1. Set the image field of the ith element in NEAREST to null
    3.2. Set the distance field of the ith element in NEAREST to infinity
4. Set smallest equal to the distance field of the kth element in NEAREST

/* Identify k closest binary images using their histograms stored in the database. */
5. For each histogram tuple, H, stored in the database
    5.1. Compute d, the distance between H and HQ.
    5.2. if d < smallest
        5.2.1. Obtain the object id associated with H and call it image
        5.2.2. Insert d and image into the NEAREST array so that the array remains
               sorted based on the distance fields.
        5.2.3. Set smallest equal to the distance field of the kth element in NEAREST

/* Determine the virtual images that satisfy the query by instantiating them and
extracting their histograms. Then, process the histograms in the same manner as for the
binary images. */
6. For each Virtual Image
    6.1. Instantiate Virtual Image
    6.2. Extract Histogram H from Instantiated Image
    6.3. Compute d, the distance between H and HQ.
    6.4. if d < smallest
        6.4.1. Obtain the object id associated with H and call it image
        6.4.2. Insert d and image into the NEAREST array so that the array remains
               sorted based on the distance fields.
        6.4.3. Set smallest equal to the distance field of the kth element in NEAREST
    6.5. Save Instantiated Image for Step 7

/* Render all retrieved images since any retrieved virtual image was converted to a
binary format in Step 6. */
7. Display first k images contained in NEAREST array
```

Figure 6-2. VSIS Algorithm for Processing Nearest Neighbor Queries

6.2. Virtual Storage with Instantiation during Insertion (VSII) Approach

The second approach in this chapter is the Virtual Storage with Instantiation during Insertion (VSII) approach, which again saves space by storing edited images virtually. This approach differs from VSIS in that it instantiates the virtual images and extracts histograms from them when they are inserted into the database. This is performed at insertion time so that the extracted histograms can be stored. After extraction, the instantiated images are discarded leaving only the virtual images and their associated histograms.

Since histograms are extracted from every image stored in the database in both approaches, the VSII algorithms for processing retrieval queries are the same as the conventional algorithms. The only difference is that the VSII algorithms may retrieve images stored virtually, which means that they must be instantiated before they can be displayed.

The VSII algorithm for processing range queries is displayed in Figure 6-3. The first three steps are used to initialize the variables and identify the parameters used by the query, and they are the same as in the VSIS and rule-based algorithms. The fourth step starts a loop that will execute once for each image in the database irrespective of how it is stored. Each iteration of the loop retrieves the current image identifier and determines if the value in the desired bin of its corresponding histogram is within the query range. The final step is to display the retrieved images which requires instantiating any virtual images that were identified in the previous step.

```
/* Identify the desired histogram bin from the parameters of the range query */
1. Initialize Results = ∅
2. Compute parameters ($PCT_{min}$, $PCT_{max}$, $C_Q$) from query
3. Compute Histogram Bin (HB) from query color $C_Q$

/* Search stored histogram bins */
4. For each histogram stored in the database
    4.1. If bin HB is within query range [$PCT_{min}$, $PCT_{max}$]
        4.1.1. Add image ID stored in tuple to Results Set

/* Render all retrieved images */
5. Display images contained in Results set
```

Figure 6-3. VSII Algorithm for Processing Range Queries

The VSII algorithm for processing nearest neighbor queries is displayed in Figure 6-4. These first four steps are the same as in the VSIS and rule-based approaches. Specifically, the first step identifies the parameters used by the query, the second step extracts a histogram from the query image, and the third and fourth steps initialize the variables used to track the k nearest distances. Step 5 of the VSII algorithm for processing nearest neighbor queries is a loop that executes once for each histogram tuple stored in the database. The algorithm computes the distance between the current histogram tuple and the histogram extracted from the query image during each pass through the loop. Since histograms are extracted for all images, both the virtual and binary images have been processed when the loop completes, so the final step of the algorithm is to display the k nearest images.

```
/* Identify the parameters of the nearest neighbor query. */
1. Identify the parameters k and Q from given query
2. Let HQ be Histogram Extracted from the Query Image Q

/* As the algorithm proceeds, the system must track the known k nearest neighbors at all
times. This information is kept in the NEAREST array. The ith element of the array
contains the identifier of the ith known closest image and its distance to Q. The following
code initializes the array. */
3. For i = 1 to k
    3.1. Set the image field of the ith element in NEAREST to null
    3.2. Set the distance field of the ith element in NEAREST to infinity
4. Set smallest equal to the distance field of the kth element in NEAREST

/* Identify k closest binary images using their histograms stored in the database. */
5. For each histogram tuple, H, stored in the database
    5.1. Compute d, the distance between H and HQ.
    5.2. if d < smallest
        5.2.1. Obtain the object id associated with H and call it image
        5.2.2. Insert d and image into the NEAREST array so that the array remains
               sorted based on the distance fields.
        5.2.3. Set smallest equal to the distance field of the kth element in NEAREST

/* Render all retrieved images */
6. Display the images contained in the first k image fields of the NEAREST array.
```

Figure 6-4. VSII Algorithm for Processing Nearest Neighbor Queries

CHAPTER 7

PERFORMANCE EVALUATION OF ALGORITHMS

This chapter presents a performance evaluation of the proposed algorithms for retrieving images in a multimedia database management system that uses virtual images. First, this chapter presents the probability that the proposed rules for retrieving virtual images will produce an error. Next, this chapter compares the proposed approaches with the conventional approach for processing retrieval queries based on in terms of storage space and execution time.

7.1. Error Probabilities

Two types of errors can occur when processing image retrieval queries. The first type is known as a *false positive*, and it occurs when the algorithm retrieves an image that does not satisfy the query. The second type is known as a *false negative*, and it occurs when an image that satisfies the query is not retrieved by the algorithm. The first type of error affects the precision of the algorithm since precision is computed as the number of retrieved and relevant images divided by the number of retrieved images [Falo, 1996]. The second type of error affects the recall of the algorithm since recall is computed as the number of retrieved and relevant images divided by the number of relevant images in the database [Falo, 1996].

Figure 7-1 illustrates a false positive and a false negative. In the figure, there are two lines with each representing the percentage of pixels [0-1] in a virtual image V that are of color C_Q. The point marked "*Actual*" represents the actual percentage of pixels that are of color C_Q in V when it is instantiated. The range marked *[PCT_{min}, PCT_{max}]* represents the range desired by the given query. The range marked "*Computed Bounds*" represents the boundary range [$BOUND_{min}$, $BOUND_{max}$], which is computed using the rules of the proposed algorithm for processing range queries. The top line in Figure 7-1 illustrates a false positive error that occurs when *Actual* does not lie within the *[PCT_{min}, PCT_{max}]* range specified by the query, but that query range does intersect the range formed by the *Computed Bounds*, which means the rule-based algorithm retrieves the image. The bottom line in Figure 7-1 illustrates a false negative error which occurs when *Actual* does lie within the *[PCT_{min}, PCT_{max}]* range specified by the query, but the query range does not intersect the range formed by the *Computed Bounds*, which means the rule-based algorithm does not retrieve the image.

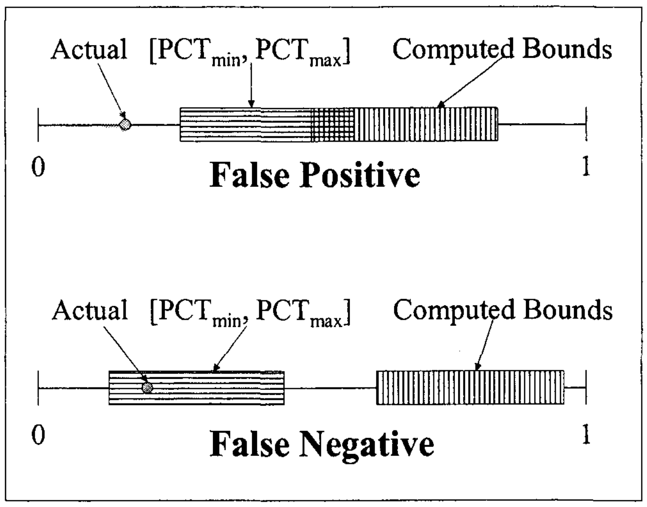


Figure 7-1. Types of Image Retrieval Errors

7.1.1. Probability of False Positives

This section will present the probability that a retrieved virtual image V is a false positive. In order to stay consistent with the variables presented in Figure 7-1, let *Query* represent the percentage range specified by the given query, which means that *Query* = [PCT_{min}, PCT_{max}]. In addition, let *Actual* represent the percentage of pixels that are of color C_Q in V when it is instantiated, and let *Bounds* represent the range formed by the maximum and minimum produced by the rules of the proposed algorithm, which means that *Bounds* = [$BOUND_{min}$, $BOUND_{max}$]. Using these variables, a false positive occurs when *Query* and *Bounds* intersect, but *Actual* $\not\subset$ *Query*.

The rule-based algorithm only returns an image when *Query* and *Bounds* intersect, which means that when an image is returned, the probability that it is a false positive is the probability that *Actual* $\not\subset$ *Query*. To calculate this value, it is necessary to define the values that *Actual* can have. Since *Actual* is a percentage, it can have any value between 0 and 1. Using this information, assume that *Actual* can be any value in that range with equal probability, meaning that it follows the uniform probability distribution. Since the possible values are between 0 and 1, the density function of *Actual, f(Actual),* is 1/(1 – 0) = 1 [Mend, 1992].

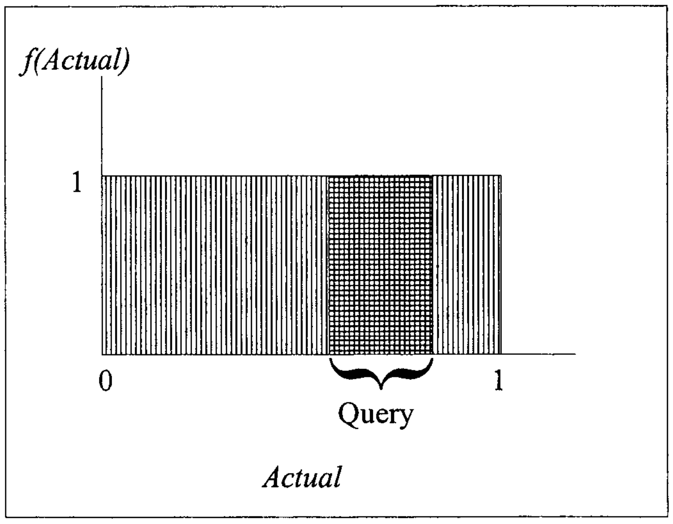


Figure 7-2. Uniform Probability Distribution Function for Actual

The probability that *Actual* $\notin$ *Query* is equal to 1 – the probability that *Actual* $\in$ *Query*. To find this probability, it is necessary to find the area under the portion of the curve marked *Actual* in Figure 7-2. The area of this rectangle equals its width since its length = 1. Since Query represents the range [PCT_{min}, PCT_{max}], the width is PCT_{max} – PCT_{min}. This means that the probability that *Actual* $\in$ *Query* is PCT_{max} – PCT_{min}, so the probability that *Actual* $\notin$ *Query* is as follows.

$$1 - (PCT_{max} - PCT_{min})$$

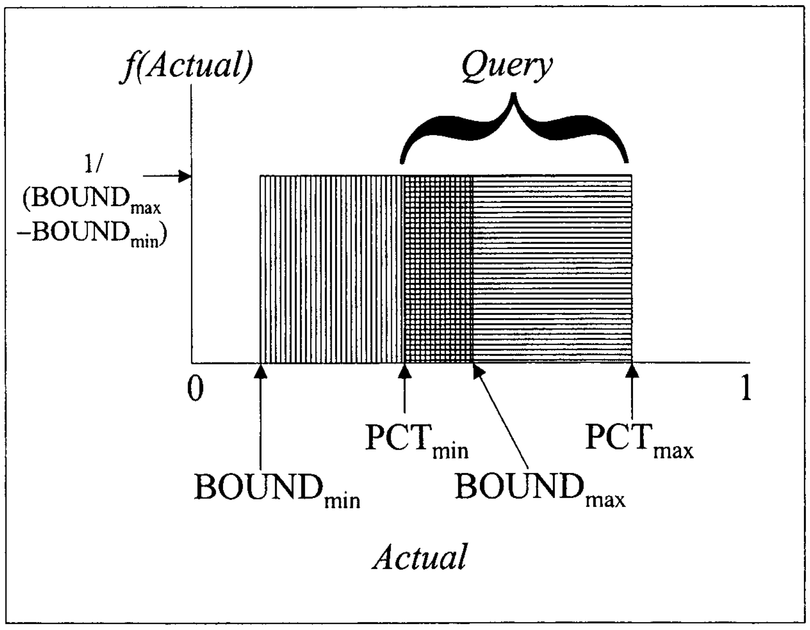


Figure 7-3. Uniform Probability Distribution Function when Actual $\in$ Bounds

The probability that *Actual* $\notin$ *Query* can be reduced by restricting the range of values that *Actual* may occupy. If it is known that *Actual* $\in$ *Bounds*, then the width of the probability distribution function for *Actual* is $BOUND_{max}$ – $BOUND_{min}$, and the height of the function is 1/[$BOUND_{max}$ – $BOUND_{min}$] [Mend, 1992] as illustrated in Figure 7-3. To compute the probability that *Actual* does not lie within *Query*, it is necessary to compute the area of the region of the probability density function that does not contain *Query*. That area is 1 – the area of the region that contains the intersection of *Query* and *Bounds*. The height of the intersection is as follows.

$$\frac{1}{BOUND_{\max} - BOUND_{\min}}$$

In addition, the width of the intersection is as follows.

$$Min(BOUND_{\max}, PCT_{\max}) - Max(BOUND_{\min}, PCT_{\min})$$

Consequently, the area of the intersection is represented by the following expression.

$$\frac{Min(BOUND_{\max}, PCT_{\max}) - Max(BOUND_{\min}, PCT_{\min})}{BOUND_{\max} - BOUND_{\min}}$$

Thus, the area of the region that does not contain *Query* is the following.

$$1 - \frac{Min(BOUND_{\max}, PCT_{\max}) - Max(BOUND_{\min}, PCT_{\min})}{BOUND_{\max} - BOUND_{\min}}$$

7.1.2. Probability of False Negatives

This section identifies the probability that a virtual image that was not retrieved by the rule-based algorithm is a false negative. As in the previous section, let *Query* represent the percentage range specified by the given query, let *Actual* represent the percentage of pixels that are of color C_Q when the virtual image is instantiated, and let *Bounds* represent the range formed by the maximum and minimum bounds produced by the rules of the proposed algorithm. Using these variables, a false negative occurs when *Query* and *Bounds* do not intersect, but $Actual \in Query$. Note that this also implies that in a false negative, $Actual \notin Bounds$.

To compute the probability that a false negative occurs, consider that although *Actual* cannot be within the range *Bounds*, it may be anywhere else within [0, 1] with equal probability. Thus, since *Bounds* is represented by the range [$\text{BOUND}_{\text{min}}$, $\text{BOUND}_{\text{max}}$], the probability distribution function for *Actual* will appear as in Figure 7-4. The width of the function is as follows.

$$1-(BOUND_{\max}-BOUND_{\min})$$

In addition, the height of the function is represented by the following expression.

$$\frac{1}{1-(BOUND_{\max}-BOUND_{\min})}$$

The probability that *Actual* is in query represents the probability that a false negative occurs, and it is equal to the area of the rectangle identified by *Query* in Figure 7-4, which is equal to the following expression.

$$(PCT_{\max}-PCT_{\min})\times(1-BOUND_{\max}+BOUND_{\min})$$

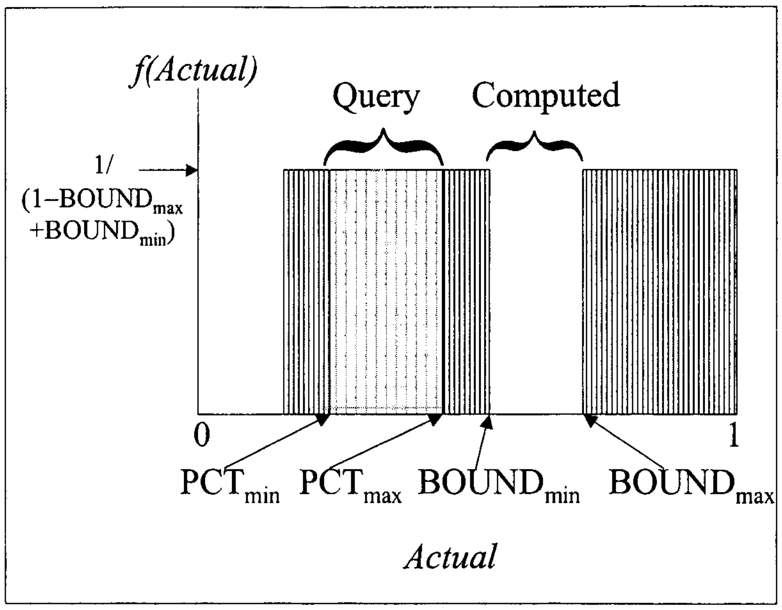


Figure 7-4. Uniform Probability Distribution Function when Actual $\notin$ Bounds

7.2. Execution Time and Storage Space

In this section, the proposed algorithms are compared to similar sequential algorithms for conventionally retrieving images using color. The conventional approach is the Binary Storage with Histograms (BSH) approach, which stores all of the images in the database in a traditional binary format and retrieves them using conventional histogram techniques. The following sections compare the approaches based on permanent storage space, image insertion time, range query processing time, and nearest neighbor query processing time. Table 7-1 displays the parameters that are used in each analysis along with their descriptions.

Parameter	Description
N	Number of Images in the Database
N_{Binary}	Number of Binary Images in the Database
$N_{Virtual}$	Number of Virtual Images in the Database
N_R	Expected Number of Retrieved Images
$N_{RBinary}$	Expected Number of Retrieved Images that are Binary
$N_{RVirtual}$	Expected Number of Retrieved Images that are Virtual
N_{Op}	Average Number of Operations within a Virtual Image
N_{Bounds}	Expected Number of Times BOUNDS is executed for Nearest Neighbor Query
T_{Range}	Average Time needed to Identify Parameters of Range Query
T_{NN}	Average Time needed to Identify Parameters of Nearest Neighbor Query
T_{Access}	Average Time needed to Access an Image
$T_{Instant}$	Average Time needed to Instantiate a Virtual Image
$T_{Display}$	Average Time needed to Display an Image
$T_{Extract}$	Average Time needed to Extract a Histogram from a Binary Image
T_{Hist}	Average Time needed to Access and Compare Histogram Bins
T_{Dist}	Average Time needed to Compute the Distance Between Two Histograms
T_{Base}	Average Time needed to Identify the Base Image of a Virtual Image
T_{Size}	Average Time needed to Identify the Number of Rows and Columns of an Image
T_{ASize}	Average Time needed to Access the Stored Numbers of Rows & Columns
T_{Bounds}	Average Time needed to Execute the BOUNDS algorithm on a virtual image
$T_{Virtual\_NN}$	Average Time needed to Execute the Virtual_NN algorithm
T_{Rule}	Average Time needed to Apply a Rule for an Editing Operation
T_{Op}	Average Time needed to Apply an Editing Operation to an Image
T_{Type}	Average Time needed to Determine if an Image is Binary or Virtual
T_{AS}	Average Time needed to Add an Image ID to the Set of Satisfying Images
T_{AB}	Average Time needed to Add a Binary Image to the Database
T_{AV}	Average Time needed to Add a Virtual Image to the Database
T_{AH}	Average Time needed to Add a Histogram to the Database
S_{Binary}	Average Size of Binary Images in the Database
$S_{Virtual}$	Average Size of Virtual Images in the Database
S_{Hist}	Average Size of Histogram Extracted From Binary Images

Table 7-1. Parameters Used in Performance Evaluation

7.2.1. Binary Storage with Histograms Approach (BSH)

The BSH algorithms are for systems that store all images in a binary format. A histogram from each image as it is inserted into the database, and the histogram is stored along with the object ID of the image. For range queries, searching is performed by accessing the extracted histograms and checking the appropriate bins representing the query color. When processing nearest neighbor queries, a histogram is extracted from the query image and compared to the histograms stored in the database to identify the ones that are the most similar.

7.2.1.1. Permanent Storage Space for BSH Algorithms

The above description implies that a system that uses the BSH algorithms to retrieve images will need space to store all of the images in a binary format and will need space to store their associated histograms. Using the variables from Table 7-1, the total space used by the binary images is $N \times S_{Binary}$, and the total space used by the histograms is $N \times S_{Hist}$. Thus, the total permanent storage space used by BSH is as follows.

$$(N \times S_{Binary}) + (N \times S_{Hist})$$

7.2.1.2. Average Insertion Time for BSH Algorithms

The BSH algorithm for inserting images is displayed in Figure 7-5. Since all images are stored in a binary format in this approach, histograms can be extracted from them. So, the first step to inserting an image is to extract its histogram from it. The next steps are to store the image and the histogram in the database. Using the variables in Table 7-1, the average time it takes to perform Step 1 is $T_{Extract}$, the average time it takes to perform Step 2 is T_{AB}, and the average time it takes to perform Step 3 is T_{AH}. So, the average total time it takes to insert an image using the algorithm is as follows.

$$T_{Extract} + T_{AB} + T_{AH}$$

```
/* Since all images are binary, extract a histogram from each image when it is inserted into the
database. */

1. Extract histogram from given image
2. Store image in database using image ID
3. Store histogram and image ID in database using Histogram ID
```

Figure 7-5. BSH Algorithm for Inserting Images

7.2.1.3. Average BSH Range Query Processing Time

The BSH algorithm for processing range queries is displayed in Figure 7-6. The first three steps are used to initialize the variables and identify the parameters used by the query, and they are the same for each of the BSH, VSIS, VSII, and rule-based algorithms. The time it takes to complete these steps is represented by the variable T_{Range}.

The fourth step starts a loop that will execute once for each image in the database, so it will execute N times. Each iteration of the loop retrieves the current image identifier and determines if the value in the desired bin of its corresponding histogram is within the query range. The average time it takes to access and compare a histogram bin to a range is represented by the variable T_{Hist}, and the average time it takes to add an image identifier to the set of retrieved images is T_{AS}. The comparison of the histogram bin will occur N times, while adding an image identifier to the set of retrieved images will occur N_R times. Thus, the total time it takes to complete Step 4 on average is as follows.

$$(N \times T_{Hist}) + (N_R \times T_{AS})$$

The final step in the BSH algorithm for processing range queries is to display the retrieved images. Since all of the images in the database are already in a binary format, the average time it takes to load and display a single image is $T_{Display} + T_{Access}$. Since N_R images are retrieved, the total average time for Step 5 is as follows.

$$N_R \times (T_{Display} + T_{Access})$$

This means that the total time it takes to execute the BSH algorithm for processing range queries is represented by the following expression.

$$(T_{Range}) + (N \times T_{Hist}) + (N_R \times T_{AS}) + (N_R \times (T_{Display} + T_{Access}))$$

```
/* Identify the desired histogram bin from the parameters of the range query */
1. Initialize Results = ∅
2. Compute parameters ($PCT_{min}$, $PCT_{max}$, $C_Q$) from query
3. Compute Histogram Bin (HB) from query color $C_Q$

/* Search stored histogram bins */
4. For each histogram stored in the database
    4.1. If bin HB is within query range [$PCT_{min}$, $PCT_{max}$]
        4.1.1. Add image ID stored in tuple to Results Set

/* Render all retrieved images */
5. Display images contained in Results set
```

Figure 7-6. BSH Algorithm for Processing Range Queries

7.2.1.4. Average BSH Nearest Neighbor Query Processing Time

The analysis for the time taken to process a nearest neighbor query considers the average time needed to identify the parameters of the query, the average time needed to extract a histogram from an image, the average time needed to compute the distance between two histograms, and the average time needed to instantiate or display an image.

Although there are other steps in the algorithm such as comparison and assignment statements, the average time used to perform those steps are considered negligible in comparison to the average times used to perform the steps listed earlier.

The BSH algorithm for processing nearest neighbor queries is displayed in Figure 7-7. These first four steps are the same for each of the BSH, VSIS, VSII, and rule-based approaches. Specifically, the first step identifies the parameters used by the query, the second step extracts a histogram from the query image, and the third and fourth steps initialize the variables used to track the k nearest distances. Since the average time it takes to initialize the variables is considered negligible, the average time it takes to complete the first four steps is $T_{nn} + T_{Extract}$.

```
/* Identify the parameters of the nearest neighbor query. */
1. Identify the parameters k and Q from given query
2. Let HQ be Histogram Extracted from the Query Image Q

/* As the algorithm proceeds, the system must track the known k nearest neighbors at all
times. This information is kept in the NEAREST array. The i^th element of the array
contains the identifier of the i^th known closest image and its distance to Q. The following
code initializes the array. */
3. For i = 1 to k
    3.1. Set the image field of the i^th element in NEAREST to null
    3.2. Set the distance field of the i^th element in NEAREST to infinity
4. Set smallest equal to the distance field of the k^th element in NEAREST

/* Identify k closest binary images using their histograms stored in the database. */
5. For each histogram tuple, H, stored in the database
    5.1. Compute d, the distance between H and HQ.
    5.2. if d < smallest
        5.2.1. Obtain the object id associated with H and call it image
        5.2.2. Insert d and image into the NEAREST array so that the array remains
               sorted based on the distance fields.
        5.2.3. Set smallest equal to the distance field of the k^th element in NEAREST

/* Render all retrieved images */
6. Display the images contained in the first k image fields of the NEAREST array.
```

Figure 7-7. BSH Algorithm for Processing Nearest Neighbor Queries

Step 5 of the BSH algorithm for processing nearest neighbor queries is a loop that executes once for each histogram tuple stored in the database, which means that it executes N times. The algorithm computes the distance between the current histogram tuple and the histogram extracted from the query image during each pass through the loop in Step 5.1. Each distance computation takes an average of T_{Dist} time, so the total time for Step 5.1 is $N \times T_{Dist}$. The remainder of the loop is considered negligible.

The final step of the above algorithm displays the k nearest images. Since each of the images in the database is stored in a binary format, the algorithm can load each image before displaying it. Thus, the total average time used to perform Step 6 is as follows.

$$N_R \times (T_{Display} + T_{Access})$$

Therefore, the average time it takes to execute the BSH algorithm for processing nearest neighbor queries is represented by the following expression.

$$(T_{NN} + T_{Extract}) + (N \times T_{Dist}) + (N_R \times (T_{Display} + T_{Access}))$$

7.2.2. Virtual Storage with Instantiation while Searching (VSIS) Approach

This section presents the analysis of the VSIS algorithms that were proposed in the previous chapter. These algorithms store derived images virtually and search them by instantiating them at the time queries are submitted to the system.

7.2.2.1. Permanent Storage Space for VSIS Algorithms

A system that uses the VSIS algorithms to retrieve images will need space to store virtual images, binary images, and histograms from the binary images. Using the variables from Table 7-1, the total space used by the virtual images is $N_{Virtual} \times S_{Virtual}$, the total space used by the binary images is $N_{Binary} \times S_{Binary}$, and the total space used by the histograms is $N_{Binary} \times S_{Hist}$. Thus, the total permanent storage space used by the VSIS algorithms is as follows.

$$(N_{Virtual} + S_{Virtual}) + (N_{Binary} \times (S_{Binary} + S_{Hist}))$$

7.2.2.2. Average Insertion Time for VSIS Algorithms

The VSIS algorithm for inserting images is displayed in Figure 7-8. If the inserted image is binary, a histogram must be extracted from it before it is added to the database. Consequently, the first step to inserting an image is to determine whether it is virtual or binary. The average time it takes to perform this step is T_{type}. If the image is binary, then the next steps are to extract its histogram, and store the image and histogram in the database. Using the variables in Table 7-1, the average time it takes to perform these steps is $T_{Extract} + T_{AB} + T_{AH}$. So, the average total time it takes to insert a binary image using the BSH algorithm is equal to the following expression.

$$T_{Extract} + T_{AB} + T_{AH}$$

Out of the N images in the database, these steps will be performed N_{Binary} times. Alternatively, if the image is virtual, the average time it takes to insert it is T_{AV}. This case will occur $N_{Virtual}$ times. Thus, the total average time it takes to execute the VSIS insertion algorithm is represented by the following expression.

$$\frac{(N \times T_{Type}) + (N_{Binary} \times (T_{Extract} + T_{AB} + T_{AH})) + (N_{Virtual} \times T_{AV})}{N}$$

```
/* Histograms should be extracted from any binary images inserted into the database, while virtual images can be inserted without extracting any values */
1. If image is binary
        1.1 Extract the histogram from given image
        1.2 Store the image in the database using its identifier
        1.3 Store the histogram and image identifier together in the database
2. Else
        2.1 Store virtual image in database using its identifier
```

Figure 7-8. VSIS Algorithm for Inserting Images

```
/* Identify the desired histogram bin from the parameters of the range query */
1. Initialize Results to ∅
2. Compute parameters ($PCT_{min}$, $PCT_{max}$, $C_Q$) from given query
3. Compute Histogram Bin (HB) from query color $C_Q$

/* Search stored histogram bins */
4. For each histogram tuple stored in the database
    4.1 If bin HB is within query range [$PCT_{min}$, $PCT_{max}$]
        4.1.1 Add image ID stored in histogram tuple to Results

/* Determine the virtual images that satisfy the query by instantiating them and extracting their histograms. Then, process the histograms in the same manner as for the binary images. */
5. For each Virtual Image
    5.1 Instantiate Virtual Image
    5.2 Extract Histogram from Instantiated Image
    5.3 If bin HB is within query range [$PCT_{min}$, $PCT_{max}$]
        5.3.1 Add image ID stored in histogram tuple to Results
        5.3.2 Save Instantiated Image for Step 6

/* Render all retrieved images since any retrieved virtual image was converted to a binary format in Step 5. */
6. Display images contained in Results set
```

Figure 7-9. VSIS Algorithm for Processing Range Queries

7.2.2.3. Average VSIS Range Query Processing Time

The VSIS algorithm for processing range queries presented previously is displayed again in Figure 7-9, and its first three steps are the same as in the BSH algorithm for processing range queries. The time it takes to complete these steps is represented by the variable T_{Range}. The fourth step is also the same as in the BSH algorithm in that it starts a loop that will execute once for each binary image in the database. Since there are only N_{Binary} images stored in the binary format, the loop will only execute N_{Binary} times. Each iteration of the loop behaves as in the BSH algorithm, so the total average time it takes to complete Step 4 is as follows.

$$(N_{Binary} \times T_{Hist}) + (N_{RBinary} \times T_{AS})$$

Step 5 of the VSIS algorithm for processing range queries starts a loop that executes once for each of the $N_{virtual}$ virtual images. Step 5.1 instantiates the current virtual image, and Step 5.2 extracts a histogram from it. The total average time to complete these steps is $T_{Instant} + T_{Extract}$. Step 5.3 is similar to Step 4.1 in that it compares the value in the desired bin to the query range, which takes an average of T_{Hist} time. Thus, the total average time it takes to complete Steps 5.1 through 5.3 is as follows.

$$N_{Virtual} \times (T_{Ins\tan t} + T_{Extract} + T_{Hist})$$

The number of virtual images that will be added to the satisfying set is represented by the variable $N_{RVirtual}$, so the total time Step 5.3.1 will use on average is $N_{RVirtual} \times T_{AS}$. Therefore, the total average time for Step 5 is equal to the following expression.

$$(N_{Virtual} \times (T_{Ins\tan t} + T_{Extract} + T_{Hist})) + (N_{RVirtual} \times T_{AS})$$

The final step in the VSIS algorithm for processing range queries is to display the retrieved images. The virtual images in the database were converted into a binary format in Step 5.2, so all of the retrieved images are in binary format. Consequently, the average time it takes to load and display the retrieved images is the same as in the BSH algorithm for processing range queries, which is as follows.

$$N_R \times (T_{Display} + T_{Access})$$

Thus, the total time it takes to execute the above VSIS algorithm is as follows.

$$(T_{Range}) + (N_{Binary} \times T_{Hist}) + (N_{RBinary} \times T_{AS}) + (N_{Virtual} \times (T_{Ins\tan t} + T_{Extract} + T_{Hist})) + \\ (N_{RVirtual} \times T_{AS}) + (N_R \times (T_{Display} + T_{Access}))$$

7.2.2.4. Average VSIS Nearest Neighbor Query Processing Time

The VSIS algorithm for processing nearest neighbor queries is displayed again in Figure 7-10, and its analysis is similar to the analysis of the BSH algorithm in that it only considers the average time needed to identify the parameters of the query, the average time needed to extract a histogram from an image, the average time needed to compute the distance between two histograms, and the average time needed to instantiate or display an image. Consequently, the total average time it takes to complete the first four steps of the VSIS algorithm is as follows.

$$T_{nn} + T_{Extract}$$

```
/* Identify the parameters of the nearest neighbor query. */
1. Identify the parameters k and Q from given query
2. Let HQ be Histogram Extracted from the Query Image Q

/* As the algorithm proceeds, the system must track the known k nearest neighbors at all
times. This information is kept in the NEAREST array. The ith element of the array
contains the identifier of the ith known closest image and its distance to Q. The following
code initializes the array. */
3. For i = 1 to k
   3.1. Set the image field of the ith element in NEAREST to null
   3.2. Set the distance field of the ith element in NEAREST to infinity
4. Set smallest equal to the distance field of the kth element in NEAREST

/* Identify k closest binary images using their histograms stored in the database. */
5. For each histogram tuple, H, stored in the database
   5.1. Compute d, the distance between H and HQ.
   5.2. if d < smallest
        5.2.1. Obtain the object id associated with H and call it image
        5.2.2. Insert d and image into the NEAREST array so that the array remains
               sorted based on the distance fields.
        5.2.3. Set smallest equal to the distance field of the kth element in NEAREST

/* Determine the virtual images that satisfy the query by instantiating them and
extracting their histograms. Then, process the histograms in the same manner as for the
binary images. */
6. For each Virtual Image
   6.1. Instantiate Virtual Image
   6.2. Extract Histogram H from Instantiated Image
   6.3. Compute d, the distance between H and HQ.
   6.4. if d < smallest
        6.4.1. Obtain the object id associated with H and call it image
        6.4.2. Insert d and image into the NEAREST array so that the array remains
               sorted based on the distance fields.
        6.4.3. Set smallest equal to the distance field of the kth element in NEAREST
   6.5. Save Instantiated Image for Step 7

/* Render all retrieved images since any retrieved virtual image was converted to a
binary format in Step 6. */
7. Display first k images contained in NEAREST array
```
Figure 7-10. VSIS Algorithm for Processing Nearest Neighbor Queries

Step 5 of the algorithm is a loop that executes once for each histogram tuple stored in the database, which means that it executes N_{Binary} times. The algorithm performs a distance computation in Step 5.1, so the average total time used to perform

Step 5.1 is $N_{Binary} \times T_{Dist}$. As in the BSH algorithm for processing nearest neighbor queries, the remainder of the loop is considered negligible.

Step 6 of the VSIS algorithm is a loop that executes once for each virtual image in the database. During each iteration, the algorithm instantiates the virtual image, extracts a histogram from it, and computes the distance between that histogram and the one extracted from the query image. Thus, each iteration through the loop requires an average of time equal to the following expression.

$$T_{Ins\tan t} + T_{Extract} + T_{Dist}$$

Consequently, the total average time required by Step 6 is represented as follows.

$$N_{Virtual} \times (T_{Ins\tan t} + T_{Extract} + T_{Dist})$$

The final step of the above algorithm displays the k nearest images. Since this algorithm instantiates all of the virtual images in Step 6, each of the retrieved images is in a binary format. Thus, the VSIS algorithm can simply load and display each binary image as with the BSH algorithm. The total average time used to perform Step 7, then is the same as in the BSH algorithm, which is as follows.

$$N_R \times (T_{Display} + T_{Access})$$

Therefore, the average time it takes to execute the VSIS algorithm for processing nearest neighbor queries is as follows.

$$(T_{Extract}) + (N_{Binary} \times T_{Dist}) + (N_{Virtual} \times (T_{Ins\tan t} + T_{Extract} + T_{Dist})) + (N_R \times (T_{Display} + T_{Access}))$$

7.2.3. Virtual Storage with Instantiation while Inserting (VSII) Approach

In this approach, the derived images are stored virtually while the base images remain stored in a binary format. This approach differs from the VSIS approach in that histograms are extracted and stored for the virtual images upon insertion which means that they must be instantiated at that time as well.

7.2.3.1. Permanent Storage Space for VSII Algorithms

The above description implies that a system that uses the VSII algorithms to retrieve images will need space to store images stored in virtual and formats as well as the histograms extracted from them. Using the variables from Table 7-1, the total space used by the virtual images is $N_{Virtual} \times S_{Virtual}$, the total space used by the binary images is $N_{Binary} \times S_{Binary}$, and the total space used by the histograms is $N \times S_{Hist}$. Thus, the total permanent storage space used by the VSII algorithms is as follows.

$$(N_{Binary} \times S_{Binary}) + (N_{Virtual} \times S_{Virtual}) + (N \times S_{Hist})$$

7.2.3.2. Average Insertion Time for VSII Algorithms

The VSII algorithm for inserting images is displayed in Figure 7-11. In order to extract a histogram from each image inserted into the database, an image stored in a virtual format must first be instantiated. So, as with the VSIS algorithm for inserting images, the first step is to determine whether it is virtual or binary, which takes an average time of T_{type}. The insertion of binary images in the VSIS algorithm is the same as in the VSII algorithm, so the average total time it takes is as follows.

$$T_{Extract} + T_{AB} + T_{AH}$$

Alternatively, if the image is virtual, the algorithm stores it, instantiates it, extracts a histogram from it, and stores the histogram. This takes a total average time as follows.

$$T_{AV} + T_{Instant} + T_{Extract} + T_{AH}$$

Binary images will be added N_{Binary} times, and virtual images will be added $N_{Virtual}$ times. Thus, the total average time it takes to execute the VSII insertion algorithm is as follows.

$$\frac{(N \times T_{Type}) + (N_{Binary} \times (T_{Extract} + T_{AB} + T_{AH})) + (N_{Virtual} \times (T_{AV} + T_{Instant} + T_{Extract} + T_{AH}))}{N}$$

```
/* Histograms should be extracted from any binary images inserted into the database */
1. If image is binary
        1.1 Extract the histogram from given image
        1.2 Store the image in the database using its identifier
        1.3 Store the histogram and image identifier together in the database

/* Virtual images must be instantiated before histograms can be extracted from them and
subsequently inserted into the database. */
2. Else
        2.1 Store virtual image in database using image ID
        2.2 Instantiate virtual image
        2.3 Extract histogram from image
        2.4 Store histogram and virtual image ID in database
```

Figure 7-11. VSII Algorithm for Inserting Images

7.2.3.3. Average Retrieval Query Processing Times for VSII Algorithms

Since histograms are extracted from every image stored in the database in both approaches, the VSII algorithms for processing retrieval queries are the same as the BSH algorithms displayed in Figures 7-6 and 7-7. The only difference is that the VSII algorithms may retrieve images stored virtually, which means that they must be instantiated before they can be displayed. Consequently, the average times it takes to execute the VSII retrieval algorithms are the same as the times it takes to execute the corresponding BSH retrieval algorithms with the exception of displaying the images.

The BSH algorithms only have to load the N_R retrieved images before displaying them, which means that require an average displaying time equal to as follows.

$$N_R \times (T_{Display} + T_{Access})$$

In the VSII algorithms, the $N_{RBinary}$ binary images can be treated in the same manner giving a total of time equal to the following expression.

$$N_{RBinary} \times (T_{Display} + T_{Access})$$

The $N_{RVirtual}$ virtual images must be instantiated before they can be displayed, which means the average time to display them is as follows.

$$N_{RVirtual} \times (T_{Display} + T_{Ins\tan t})$$

Thus, the total average time needed to display all of the retrieved images in the VSII retrieval algorithms is equal to the following expression.

$$N_{RBinary} \times ((T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t}))$$

This implies that the VSII range query processing algorithm takes an average of time equal to the following expression.

$$(T_{Range}) + (N \times T_{Hist}) + (N_R \times T_{AS}) + (N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t}))$$

In addition, the VSII nearest neighbor query processing algorithm takes an average of time equal to the next expression.

$$(T_{NN} + T_{Extract}) + (N \times T_{Dist}) + (N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t}))$$

7.2.4. Rule-Based Approach

This section presents the analysis of the rule-based algorithms that were proposed in the previous chapters. These algorithms use histograms to search the binary images that are stored in the database and the proposed rules to search the virtual images.

7.2.4.1 Permanent Storage Space for Rule-Based Algorithms

In the rule-based algorithms, the N_{Binary} binary images are processed in the same manner as the BSH algorithms, which means that histograms are extracted from each of them. Thus, the total space used for the binary images is as follows.

$$(N_{Binary} \times S_{Binary}) + (N_{Binary} \times S_{Hist})$$

The $N_{Virtual}$ virtual images are processed using their descriptions, so it uses $N_{Virtual} \times S_{Virtual}$ space. In addition, the algorithms require storing the numbers of rows and columns of each binary image in the database, so the rule-based algorithms require an extra $2 \times N_{Binary}$ space. Thus, the total permanent storage space required by the rule-based algorithms is as follows.

$$(N_{Binary} \times (S_{Binary} + S_{Hist} + 2)) + (N_{Virtual} \times S_{Virtual})$$

7.2.4.2 Average Insertion Time for Rule-Based Algorithms

The rule-based algorithm for inserting images is displayed in Figure 7-12, and it is similar to the VSIS algorithm for inserting images. The difference is that the numbers of rows and columns in a binary image must be identified and stored when one is inserted into the database. The average time it takes to identify these values is represented by the variable T_{Size}. The average time to insert a binary image in the VSIS algorithm is as follows.

$$T_{Extract} + T_{AB} + T_{AH}$$

Also, the average time to insert a binary image in the rule-based algorithm is as follows.

$$T_{Extract} + T_{AB} + T_{AH} + T_{Size}$$

Consequently, the total average time it takes to execute the rule-based insertion algorithm is equal to the following expression.

$$\frac{(N \times T_{Type}) + (N_{Binary} \times (T_{Extract} + T_{AB} + T_{AH} + T_{Size})) + (N_{Virtual} \times T_{AV})}{N}$$

```
/* Histograms should be extracted from any binary images inserted into the database */
1. If image is binary
        1.1 Extract the histogram from given image
        1.2 Extract and store the numbers of rows and columns for use by BOUNDS
        1.3 Store the image in the database using its identifier
        1.4 Store the histogram and image identifier together in the database

/* Virtual images are inserted into the database correctly. */
2. Else
        2.1 Store virtual image in database using image ID
```

Figure 7-12. Rule-Based Algorithm for Inserting Images

7.2.4.3. Average Rule-Based Range Query Processing Time

The rule-based algorithm for processing range queries was presented in Chapter 3. The first three steps of the algorithm identified the parameters of the query and initialized the variables, which takes an average of T_{Range} time. Step 4 of the algorithm was the same as Step 4 of the VSIS algorithm for processing Range queries, which means that the average time to complete it will be the same as the following time of the VSIS algorithm.

$$(N_{Binary} \times T_{Hist}) + (N_{RBinary} \times T_{AS})$$

Step 5 of the algorithm executed the BOUNDS procedure for each virtual image in the database. The average time to execute the BOUNDS procedure is represented by the variable T_{Bounds}, which means the total average time executing the procedure will be $N_{Virtual} \times T_{Bounds}$. In addition, $N_{RVirtual}$ images are expected to be retrieved, and that will take an additional $N_{RVirtual} \times T_{AS}$ time.

The last step of the rule-based range query processing algorithm displays the retrieved images. As with the VSII algorithm, the virtual images must be instantiated before they can be displayed. Consequently, the average time needed to display the retrieved images in the rule-based algorithm is the same as in the VSII algorithm, which is as follows.

$$(N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t}))$$

So, the total average time needed to execute the rule-based algorithm for processing range queries is represented by the following expression.

$$(T_{Range}) + (N_{Binary} \times T_{Hist}) + (N_{RBinary} \times T_{AS}) + (N_{Virtual} \times T_{Bounds}) + (N_{RVirtual} \times T_{AS}) +$$
$$(N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t}))$$

7.2.4.4. Average Rule-Based Nearest Neighbor Query Processing Time

The rule-based algorithm for processing nearest neighbor queries was presented in Chapter 4. The first four steps of the algorithm are the same as in the previous algorithms for the BSH, VSIS, and VSII approaches, and they take an average of $T_{nn} + T_{Extract}$ time. Step 5 of the algorithm is a loop that is the same as in the VSIS algorithm. Thus, the average total time used to complete the loop is $N_{Binary} \times T_{Dist}$, as in the VSIS algorithm. Step 6 of the rule-based algorithm executes the VIRTUAL_NN procedure, which takes an average time of $T_{Virtual\_NN}$. The last step of the rule-based algorithm displays the retrieved images, which again requires an average of time as follows.

$$(N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t}))$$

Thus, the total average time used to execute the rule-based algorithm is represented by the following expression.

$$T_{NN} + T_{Extract} + (N_{Binary} \times T_{Dist}) + T_{Virtual\_NN} +$$
$$(N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t}))$$

7.3. Comparison of Approaches

This section compares the performance of each of the above approaches for retrieving virtual images. The comparisons are performed using relationships between the variables in Table 7-1. For example, since each image must be stored in either a virtual or a binary format, $N = N_{Binary} + N_{Virtual}$, and $N_R = N_{RBinary} + N_{RVirtual}$. In addition, this section assumes that the average size of a binary image is much larger than the average size of a virtual image, so $S_{Binary} > S_{Virtual}$. Finally, this section assumes that the technique used to instantiate a virtual image involves accessing the base image stored within the first line of its description, then applying each of the associated editing operations. Thus, the average time used to instantiate a virtual image, $T_{Instant}$, can also be represented by the following expression.

$$T_{Access} + (N_{Op} \times T_{Op})$$

7.3.1. Comparison of Permanent Storage Space

Table 7-2 compares the total amount of storage space used by the various approaches. The table decomposes the terms presented earlier into the amount of space used to store binary images, the amount of space used to store virtual images, the amount of space used to store the extracted histograms, and the amount of space used to store the numbers of rows and columns of the binary images. These totals are represented by the "Binary Images", "Virtual Images", "Histograms", and "Sizes" columns, respectively.

Approach	**Binary Images**	**Virtual Images**	**Histograms**	**Sizes**
BSH	$N \times S_{Binary}$	0	$N \times S_{Hist}$	0
VSIS	$N_{Binary} \times S_{Binary}$	$N_{Virtual} \times S_{Virtual}$	$N_{Binary} \times S_{Hist}$	0
VSII	$N_{Binary} \times S_{Binary}$	$N_{Virtual} \times S_{Virtual}$	$N \times S_{Hist}$	0
Rule-Based	$N_{Binary} \times S_{Binary}$	$N_{Virtual} \times S_{Virtual}$	$N_{Binary} \times S_{Hist}$	$N_{Binary} \times 2$

Table 7-2. Comparison of Total Space Used by Each Approach

Since S_{Binary} is expected to be much larger than $S_{Virtual}$, the BSH algorithms are expected to use much more space than the BSH, VSIS, and VSII algorithms. This is because the BSH algorithms store all images in a binary format while the other algorithms store $S_{Virtual}$ of the images virtually. In addition, since $N > N_{Binary}$, the following expression is true.

$$(N \times S_{Hist}) > (N_{Binary} \times S_{Hist})$$

The above expression also means that the BSH and VSII algorithms are expected to use more space to store the histograms extracted from images than the rule-based and VSIS algorithms.

When comparing the VSIS, VSII, and rule-based algorithms, all three are expected to use the same space to store the images. The VSII algorithms, however, are expected to use the following amount of additional space to store the histograms than the other two approaches.

$$(N \times S_{Hist}) - (N_{Binary} \times S_{Hist})$$

The above expression can be simplified to $N_{Virtual} \times S_{Hist}$. The rule-based algorithms are expected to use $2 \times N_{Binary}$ more space than the other approaches to store the numbers of rows and columns in each binary image. Thus, the VSIS algorithms are expected to use the least space, followed by either the rule-based algorithms or the VSII algorithms depending on whether $N_{Virtual} \times S_{Hist}$ or $2 \times N_{Binary}$ is smaller.

7.3.2. Comparison of Average Insertion Time

Table 7-3 compares the average insertion time produced by each of the four approaches. The table decomposes the times presented earlier into the average time needed to determine whether an image is virtual or binary, the average time needed to insert a binary image, and the average time needed to insert a virtual image for each approach. These times are represented by the "Type of Image", "Binary Images", and "Virtual Images" columns, respectively.

Approach	**Type of Image**	**Binary Images**	**Virtual Images**
BSH	0	$T_{Extract} + T_{AB} + T_{AH}$	0
VSIS	T_{type}	$T_{Extract} + T_{AB} + T_{AH}$	T_{AV}
VSII	T_{type}	$T_{Extract} + T_{AB} + T_{AH}$	$T_{AV} + T_{Instant} + T_{Extract} + T_{AH}$
Rule-Based	T_{type}	$T_{Extract} + T_{AB} + T_{AH}$	T_{AV}

Table 7-3. Comparison of Average Insertion Times for Each Approach

When comparing the average insertion times for each approach, assume that the values $T_{Extract}$ and $T_{Instant}$ should be much larger than the values T_{AV}, T_{AB}, T_{AH}, and T_{type}. Thus, the latter values are assumed to be negligible. Since all N images are binary in the BSH algorithms, the total time for insertion should be $N \times T_{Extract}$. Thus, the average insertion time should be $T_{Extract}$.

Alternatively, the total insertion time for the VSII approach should be as follows.

$$(N_{Binary} \times T_{Extract}) + (N_{Virtual} \times (T_{Ins\tan t} + T_{Extract}))$$

This implies that the average insertion time should be equal to the following expression.

$$\frac{N_{Binary} \times T_{Extract}}{N} + \frac{N_{Virtual} \times (T_{Ins\tan t} + T_{Extract})}{N}$$

The above expression can be simplified as follows:

$$\left(\frac{N_{Binary}}{N} \times T_{Extract}\right) + \left(\frac{N_{Virtual}}{N} \times (T_{Ins\tan t} + T_{Extract})\right)$$

$$\frac{N_{Binary} + N_{Virtual}}{N} \times T_{Extract} + \frac{N_{Virtual}}{N} \times T_{Ins\tan t}$$

$$T_{Extract} + (\frac{N_{Virtual}}{N} \times T_{Ins\tan t})$$

Since the above expression is greater than $T_{Extract}$, the average time needed to insert an image using the VSII algorithms is longer than the corresponding time for the BSH algorithms.

The average times to insert an image using the VSIS and rule-based algorithms are the same, according to Table 7-3. Since the total insertion time is expected to be $N_{Binary} \times T_{Extract}$, the average insertion time should be as follows.

$$\frac{N_{Binary}}{N} \times T_{Extract}$$

Thus, the average time used to insert an image using the VSIS and rule-based algorithms is expected to be shorter than the other algorithms.

7.3.3. Comparison of Average Times for Processing Range Queries

Table 7-4 compares the average time used to process a range query using the four approaches. The "Param" column represents the average amount of time used to identify the parameters of the range query, the "Hist" column represents the average amount of time used to access a histogram bin and compare it to the query range, the "Results" column represents the average amount of time used to add an image ID to the results set, the "Display Binary" column represents the average amount of time used to display a binary image, and the "Display Virtual" column represents the average amount of time used to display a virtual image.

Approach	Param	Hist	Virtual	Results	Display Binary	Display Virtual
BSH	T_{Range}	$N \times T_{Hist}$	0	$N_R \times T_{AS}$	$N_R \times (T_{Access} + T_{Display})$	0
VSIS	T_{Range}	$N_{Binary} \times T_{Hist}$	$N_{Virtual} \times (T_{Instant} + T_{Extract})$	$N_R \times T_{AS}$	$N_{RBinary} \times (T_{Access} + T_{Display})$	$N_{RVirtual} \times (T_{Access} + T_{Display})$
VSII	T_{Range}	$N_{Binary} \times T_{Hist}$	$N_{Virtual} \times T_{Hist}$	$N_R \times T_{AS}$	$N_{RBinary} \times (T_{Access} + T_{Display})$	$N_{RVirtual} \times (T_{Access} + T_{Instant})$
Rule-Based	T_{Range}	$N_{Binary} \times T_{Hist}$	$N_{Virtual} \times T_{Bounds}$	$N_R \times T_{AS}$	$N_{RBinary} \times (T_{Access} + T_{Display})$	$N_{RVirtual} \times (T_{Access} + T_{Instant})$

Table 7-4. Comparison of Average Times for Processing Range Queries

The time used to identify the parameters and the time used to retrieve an image ID are not included in the comparison since they are the same for all four approaches. The time used to access and compare a histogram bin to the query value should be small, so those values should be considered negligible, which means that the "Hist" column is not included in the comparison. Thus, when comparing approaches, consider only the columns for processing virtual images and displaying images.

When comparing how long it takes the query processor is to identify the images that satisfy a given query for each approach, then it is not necessary to compare the time it takes to display an image. Given that assumption, the only column contributing to the comparison is the average time taken to process virtual images. The VSII range query processing algorithm checks histograms for the virtual images, so its average time should be the same as the time for the corresponding BSH algorithm. The average time used by the rule-based algorithm is based on the average time needed to execute the BOUNDS algorithm, which is represented by the following expression.

$$T_{Base} + T_{Size} + (N_{Op} \times T_{Rule})$$

The average time used by the VSIS algorithm is based on the average time needed to instantiate an image, which is as follows.

$$T_{Base} + T_{Access} + (N_{Op} \times T_{Op})$$

Assuming that the variables T_{Size} and T_{Access} are comparable, and T_{Op} should be much larger than T_{Rule}, the average time for executing the VSIS algorithm should be much larger than the rule-based algorithm.

7.3.4. Comparison of Average Times for Processing Nearest Neighbor Queries

Table 7-5 compares the average time used to process a nearest neighbor query using the four approaches. The "Extraction" column represents the average amount of time used to extract a histogram from the Query Image, the "Binary" column represents the average amount of time used to compute the distance between the query histogram and the histograms extracted from the binary images, the "Virtual" column represents the average amount of time process the virtual images, the "Display Binary" column represents the average amount of time used to display a binary image, and the "Display Virtual" column represents the average amount of time used to display a virtual image.

Approach	Extraction	Binary	Virtual	Display Binary	Display Virtual
BSH	$T_{Extract}$	$N \times T_{Dist}$	0	$N_R \times (T_{Access} + T_{Display})$	0
VSIS	$T_{Extract}$	$N_{Binary} \times T_{Dist}$	$N_{Virtual} \times (T_{Instant} + T_{Extract} + T_{Dist})$	$N_{RBinary} \times (T_{Access} + T_{Display})$	$N_{RVirtual} \times (T_{Access} + T_{Display})$
VSII	$T_{Extract}$	$N_{Binary} \times T_{Dist}$	$N_{Virtual} \times T_{Dist}$	$N_{RBinary} \times (T_{Access} + T_{Display})$	$N_{RVirtual} \times (T_{Access} + T_{Instant})$
Rule-Based	$T_{Extract}$	$N_{Binary} \times T_{Dist}$	$T_{Virtual\_NN}$	$N_{RBinary} \times (T_{Access} + T_{Display})$	$N_{RVirtual} \times (T_{Access} + T_{Instant})$

Table 7-5. Comparison of Average Times for Processing Nearest Neighbor Queries

The time used to extract the histogram from the query image is not included in the comparison since it is the same for all four approaches. As with the comparison for the range query, if the goal of the algorithm is to identify the nearest images to a given query image, it is not necessary to include the average time used to display the images. Thus, the comparison between the four approaches is based on the columns for processing the binary and virtual images.

Both the BSH and VSII algorithms are expected to take the same time to process a nearest neighbor query, which is N x T_{Dist}. The VSIS algorithm is expected to use time equal to the following expression.

$$(N_{Binary} \times T_{Dist}) + (N_{Virtual} \times (T_{Ins\tan t} + T_{Extract} + T_{Dist}))$$

The above expression simplifies as follows.

$$(N \times T_{Dist}) + (N_{Virtual} \times (T_{Ins\tan t} + T_{Extract}))$$

Thus, the average time to execute the VSIS algorithm for processing nearest neighbor queries should be longer than the times for the BSH and VSII algorithms.

The expected time used by the rule-based algorithm to process nearest neighbor queries is represented by the following expression.

$$(N_{Binary} \times T_{Dist}) + (T_{Virtual\_NN})$$

The expected time used by the BSH and VSII algorithms can be expressed as follows.

$$(N_{Binary} \times T_{Dist}) + (N_{Virtual} \times TDist)$$

Thus, the comparison of the approaches depends on whether it takes longer to execute the VIRTUAL_NN procedure or make $N_{Virtual}$ distance computations.

7.3.5. Summary of Comparisons

The above comparisons demonstrate the rule-based algorithms provide many benefits when considering permanent storage space, insertion time, and retrieval time. By storing editing images virtually, the rule-based algorithms use less permanent storage space than the BSH algorithms. Because they avoid instantiating the virtual images, the rule-based algorithms are relatively fast in inserting virtual images unlike the VSII algorithms. For the same reason, the rule-based algorithms are relatively fast in processing retrieval queries unlike the VSIS algorithms.

7.4. Analysis of Proposed Data Structure

A data structure is presented in Chapter 5 that can be used to reduce the time it takes to process a range query using the rules proposed in Chapter 3. The purpose of this section is to analyze the algorithms presented for the data structure in order to compare their performance against the performance of the proposed rule-based algorithms. Table 7-6 lists additional variables that are used in the performance evaluation.

Parameter	Description
N_{Main}	Number of virtual images that contain only operations with bound-widening rules
$N_{Unclass}$	Number of virtual images that have an operation whose rule is not bound-widening
T_{Main}	Average Time needed to Access an Element of the Main Component
$T_{Unclass}$	Average Time needed to Access an Element of the Unclassified Component
$T_{AddMain}$	Average Time needed to Add an Element to the Main Component
$T_{AddUnclass}$	Average Time needed to Add an Element to the Unclassified Component
S_{ID}	Average Size of an Identifier

Table 7-6. Additional Variables Used in Evaluation of Data Structure

7.4.1. Permanent Storage Space for Proposed Data Structure

The proposed data structure is to be used by the underlying database management system along with the images and histograms. Thus, utilizing the proposed data structure will result in using more space than is needed by the rule-based algorithms. Specifically, the permanent storage space is equal to the storage space used by the rule-based algorithms added to the space needed to store the data structure.

The permanent storage space required by the rule-based algorithms is as follows.

$$(N_{Binary} \times (S_{Binary} + S_{Hist} + 2)) + (N_{Virtual} \times S_{Virtual})$$

To determine the space needed by the proposed data structure, consider that the Main Component will contain exactly one histogram identifier for each binary image in the database. In addition, the identifier of each virtual image will be added to either the Main Component or the Unclassified Component, which means that each identifier will be added exactly once to the data structure. Thus, the data structure will contain a total of N identifiers, which means that the total amount of permanent storage space expected to be used by the algorithms for the proposed data structure is as follows.

$$(N_{Binary} \times (S_{Binary} + S_{Hist} + 2)) + (N_{Virtual} \times S_{Virtual}) + (N \times S_{ID})$$

7.4.2. Average Insertion Time for Proposed Data Structure

Since the purpose of the proposed data structure is to arrange the image identifiers in order to speed up processing of the retrieval queries, it is expected that the average time to insert into a system that utilizes the data structure will be longer than the average time to insert into a system that does not utilize it. This section presents an analysis of the time it takes to insert an ID into the proposed data structure.

To insert an ID for an image into the proposed data structure, it is first necessary to determine the type of image it references, which should take T_{type} time. If the image is binary, the identifier of its histogram is added to the end of the Main Component, which should take $T_{AddMain}$ time. If the image is virtual, the insertion algorithm presented in Chapter 5 is executed.

The first two steps of the insertion algorithm identify the base image and its associated histogram, which takes T_{Base} time. The third step is a loop that executes for each editing operation within the description of the virtual image. During each step, the rule for the editing operation is checked to determine if it is bound-widening. The average time taken for this step can be represented by the variable T_{Rule}, so the total time used by this step is expected to be $N_{Op} \times T_{Rule}$. If the loop identified a rule that was not bound-widening, the final step of the algorithm is to add the identifier of the virtual image the Unclassified Component, which takes $T_{AddUnclass}$ time. Alternatively, if all of the rules were bound-widening, the final step is to add the identifier to the Main Component in the list of the appropriate histogram identifier. Adding the identifier should take $T_{AddMain}$ time, while identifying the histogram identifier should take the following time.

$$\frac{N_{Binary}}{2} \times T_{Main}$$

Out of N images, the insertion of a binary image should occur N_{Binary} times, and the insertion of a virtual image should occur $N_{Virtual}$ times. So, the average time needed to insert an identifier into the proposed data structure is equal to the following expression where α is the average time to insert a virtual image.

$$T_{type} + \frac{N_{Binary} \times T_{AddMain}}{N} + \frac{N_{Virtual} \times \alpha}{N}$$

A total of N_{Main} of the virtual images will be added to the Main Component, while $N_{Unclass}$ of them will be added to the Unclassified Component. So, α equals the next expression.

$$T_{Base} + (N_{Op} \times T_{Rule}) + \frac{N_{Main}}{N_{Virtual}} \times \left(T_{AddMain} + (\frac{N_{Binary}}{2} \times T_{Main}) \right) + \frac{N_{Unclass}}{N_{Virtual}} \times (T_{AddUnclass})$$

7.4.3. Average Range Query Processing Time for Proposed Data Structure

The algorithm for processing a range query using the proposed data structure has the same first three steps as the previously presented algorithms. Thus, the average time needed to execute them is T_{Range}. Step 4 accesses each element in the Unclassified Component and executes the BOUNDS algorithm. Thus, the total time expected to be taken by Step 4 is $N_{Unclass} \times T_{Bounds}$. In addition, during the execution of this query processing algorithm, the same number of images should be retrieved as in the rule-based algorithm. Thus, the total time used to add image identifiers to the set of satisfying images will be $N_R \times T_{AS}$.

Step 5 accesses each of the N_{Binary} histograms in the Main Component. Steps 5.1 and 5.2 access the desired histogram bin and compare it to the query range, which takes T_{Hist} time. This implies that Steps 5.1 and 5.2 are expected to take a total of ($N_{Binary} \times T_{Hist}$) time.

If the histogram bin falls within the query range, which should occur $N_{RBinary}$ times, all of the elements in V_list are retrieved. If the histogram bin is not within the query range, which should occur $N_{Binary} - N_{RBinary}$ times, the BOUNDS algorithm is executed for each element in the list. Since there are N_{Binary} binary images and N_{Main} virtual images represented in the Main Component, each V_list is expected to have N_{Main}/N_{Binary} virtual image identifiers. Thus, the BOUNDS algorithm will be executed the following number of times.

$$(N_{Binary} - N_{RBinary}) \times \frac{N_{Main}}{N_{Binary}}$$

The final step of the algorithm for processing range queries using the proposed data structure displays the retrieved images. This is expected to take the same amount of time as in the rule-based algorithm since the images in the database are stored in the same format. Thus, the average time for displaying the retrieved images is as follows.

$$(N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t}))$$

To summarize, the average time used to process a range query using the proposed data structure is equal to the following expression.

$$T_{Range} + (N_{Unclass} \times T_{Bounds}) + (N_{Binary} \times T_{Hist}) + ((N_{Binary} - N_{RBinary}) \times \frac{N_{Main}}{N_{Binary}} \times T_{Bounds}) +$$
$$(N_R \times T_{AS}) + (N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t})).$$

This is in comparison to the average time used by the rule-based algorithm to process a range query, which is as follows.

$$(T_{Range}) + (N_{Binary} \times T_{Hist}) + (N_R \times T_{AS}) + (N_{Virtual} \times T_{Bounds}) +$$
$$(N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Ins\tan t}))$$

Eliminating the terms (T_{Range}), ($N_{Binary} \times T_{Hist}$), ($N_R \times T_{AS}$), and ($N_{RBinary} \times (T_{Display} + T_{Access})) + (N_{RVirtual} \times (T_{Display} + T_{Instant})$) from both expressions yields the following values representing the average time used for the data structure and the average time used by the rule-based algorithm, respectively.

$$(N_{Unclass} \times T_{Bounds}) + ((N_{Binary} - N_{RBinary}) \times \frac{N_{Main}}{N_{Binary}} \times T_{Bounds})$$

$$N_{Virtual} \times T_{Bounds}$$

.

The average time used for the proposed data structure can be simplified in the following manner.

$$(N_{Unclass} + (N_{Binary} - N_{RBinary}) \times \frac{N_{Main}}{N_{Binary}}) \times T_{Bounds}$$
$$(N_{Unclass} + N_{Main} - (N_{RBinary} \times \frac{N_{Main}}{N_{Binary}}) \times T_{Bounds}$$

Let ($N_{RBinary} \times N_{Main}/N_{Binary}$) be represented by the variable β, and note that $\beta > 0$. Since $N_{Virtual}$ equals $N_{Unclass} + N_{Main}$, the average time used for the proposed data structure is equivalent to the following expression.

$$(N_{Virtual} - \beta) \times T_{Bounds}$$

Since the average time used by the rule-based algorithm is $N_{Virtual} \times T_{Bounds}$, it is greater than the average time used by the proposed data structure.

CHAPTER 8

A PROTOTYPE VIRTUAL IMAGE RETRIEVAL SYSTEM

In order to confirm the analysis presented in the previous chapter, several prototype image retrieval systems were developed during this research. This chapter describes the implementation of these systems including the development environment and the resulting user interface along with a diagram of the information flowing between the modules of the systems. In addition, this chapter provides a performance evaluation of the prototype systems that illustrate their various strengths and weaknesses.

The structure of the remainder of the chapter is as follows. Section 8.1 describes the implementation of the prototypes virtual image retrieval systems. Section 8.2 describes the performance parameters used in this research, and Section 8.3 presents the results of processing range and nearest neighbor queries with the prototypes.

8.1. Implementation

The prototype virtual image retrieval system [Brow, 2001] developed as a result of this dissertation is a web-enabled prototype accessible from the URL http://www.cs.ou.edu/~lbrown. The system processes the retrieval queries on the web server on the network of the School of Computer Science at The University of Oklahoma and sends the results as a web page to the client of the user. It was developed using the Perl language on a SUNsparc workstation using the Unix operating system, and does not use any commercial software for managing the databases. It does use utilities from the pbmplus [PBMP, 2003] package, however, to convert binary images between the text-based ppm format and more commonly used formats such as gif and jpeg.

The prototype, called the Virtual image Retrieval System (VRS), implements the rule-based algorithms presented in Chapters 3 and 4. In addition, additional prototypes were constructed in order to implement the alternative approaches to retrieving virtual images. One prototype stores all of the images in a binary format and implements the conventional histogram algorithms. This prototype is referred to as the Binary Storage with Histograms prototype (BSH). Another prototype, which is called the Virtual Storage with Instantiation while Searching Prototype (VSIS), stores edited images virtually and instantiates them during retrieval query processing in order to utilize histograms. The final prototype, called the Virtual Storage with Instantiation while Inserting (VSII) Prototype, stores edited images virtually but instantiates them at the time they are inserted in the database in order to extract their histograms. Afterwards, the histograms are stored in the database while the instantiated version of the virtual images is discarded.

8.1.1. Prototype Structure

Figure 8-1 displays the components of the VRS and illustrates how they interact. Specifically, the figure contains the model for a DataBase Management System (DBMS) that uses virtual images. In the figure, users interact with various interfaces in order to perform the different tasks common to DBMSs. When users enter an image stored in a binary format into the system through the insertion interface, it must be sent to the feature extraction module before it is stored in the underlying database. The feature extraction module extracts the color histogram from the binary image and stores it in the database as well. When users enter a virtual image into the database, they will have to access an editing interface that allows them to add or remove image editing operations. To summarize, the DBMS must provide interfaces to allow users to interact with the three types of data items stored in the database, images represented using a conventional format, values of features extracted from those images, and images stored virtually.

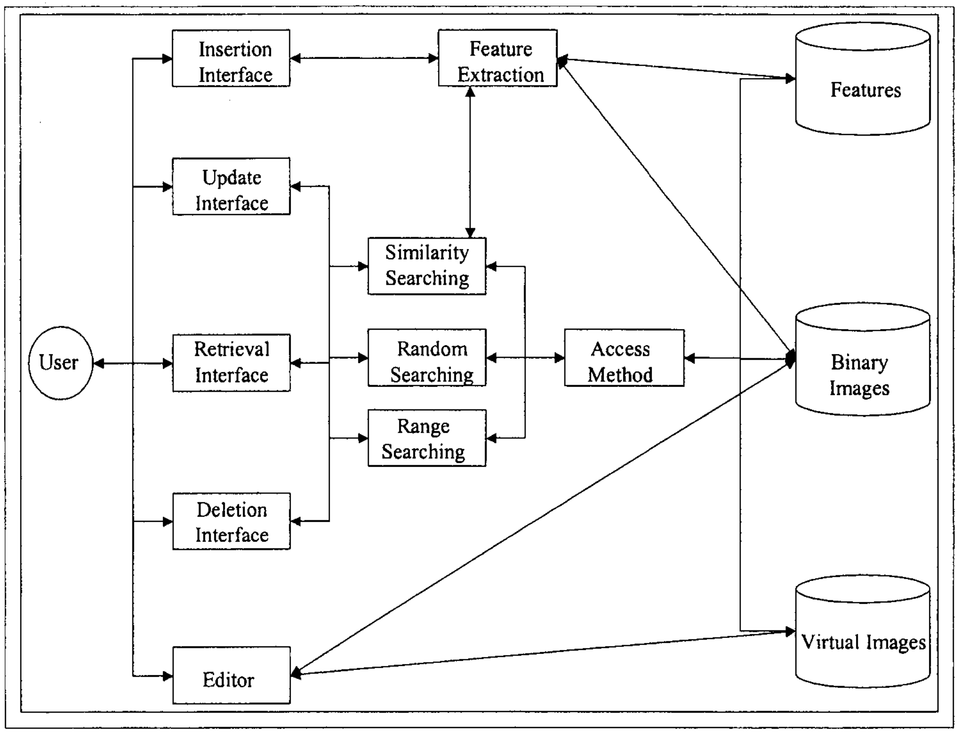


Figure 8-1. Components of the Virtual Image Retrieval System Prototype

When users want to update or delete an image from the system, they must first identify the image through the update or deletion interface. Once the image is identified, it must be located in the database using the random searching module. The module may utilize an access method such as an index in order to make the searching more efficient.

The image may be stored in a conventional format or stored virtually, so the access method must be able to access both data sets.

When the user submits a query through the retrieval interface, the DBMS should locate those images that satisfy it. The query may be a random query in which a specific image is requested, a range query in which images that have features within certain values are requested, or a similarity search which requests the images that are similar to a given query image. To locate the images that satisfy these queries, the searching modules must be able to access the virtual images, conventional images, and features. Again, these modules should use some type of access method in order to make searching more efficient. In addition, in order to determine if the images in the database are similar to a query image, the similarity search module will need to send the query image to the feature extraction module.

8.1.2. Queries and Images

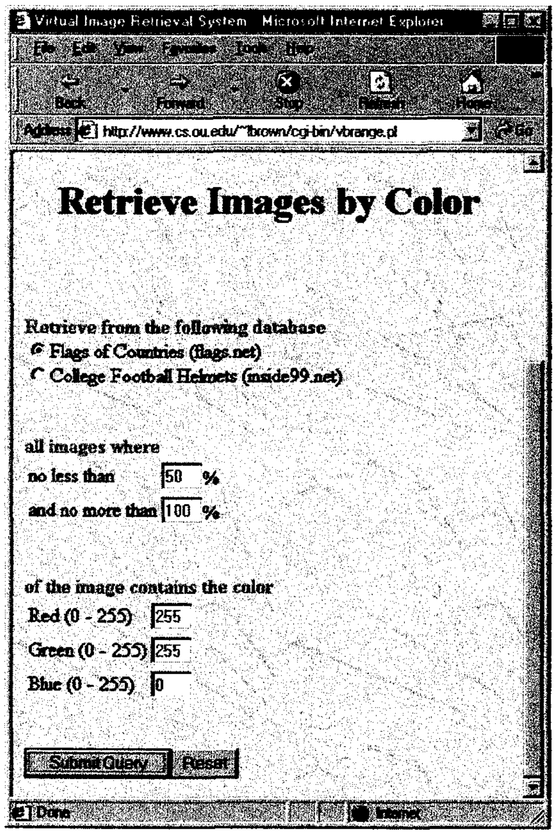
The VRS prototype allows users to retrieve images using the two different types of queries described in Chapters 3 and 4 of this dissertation. The first type is the range query "*Retrieve all images that are between PCT_{min} and PCT_{max} percent of color C_Q*", where $\mathrm{PCT}_{\mathrm{min}}$ and $\mathrm{PCT}_{\mathrm{max}}$ represent percentages and C_{Q} represents a color in the RGB model. For example, if $\mathrm{PCT}_{\mathrm{min}}$ is 10, $\mathrm{PCT}_{\mathrm{max}}$ is 100, and C_{Q} is (255,0,0), then the query is to retrieve all images that are between 10% and 100% of color (255,0,0), which is equivalent to retrieving all images that are at least 10% red. The second type of query is a nearest neighbor of the form "*Retrieve the k images that most resemble Q based on color*", where k represents a number, and Q represents a query image.

The prototype retrieves images from data sets obtained from various sites on the Internet ([Flag, 2003], [Helm, 2003]). Each data set consists of images stored as gif files. The first data set contains a collection of images of flags around the world [Flag, 2003], and the second contains a collection of images of college football helmets [Helm, 2003]. Many of the images in the flag data set are very similar, such as the flags of France and Italy. In addition, some of the flag images are closer views of portions of other flag images. In the collection of images from the Helmet Project, many images only differ in the color of the facemask or logo. Again, this means that several of the images in that data set are similar.

8.1.3. User Interface

To submit the range query described in the previous section to the VRS, users must access the screen displayed in Figure 8-2. This screen allows the user to populate a form by entering values for PCT_{min}, PCT_{max}, and C_Q. The values for PCT_{min} and PCT_{max} are restricted to integers between 0 and 100. The color C_Q is expressed as a value in the RGB color model, so the user must enter a value between 0 and 255 for each of the Red, Green, and Blue axes of that model. In addition, the interface requires the user specify the data set to search since there are multiple data sets in the system.

Upon submission of the form, the prototype executes the rule-based range query processing algorithm. Upon completion of the algorithm, the system generates and displays a web page back to the user containing the set of retrieved images. Specifically, the system displays the thumbnail of each retrieved image along with its associated filename in the web page. An example of such a page is shown in Figure 8-3.


Virtual Image Retrieval System - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.cs.ou.edu/~tbrown/cgi-bin/vbrange.pl
Go
Submit Query
Reset
Done
Internet

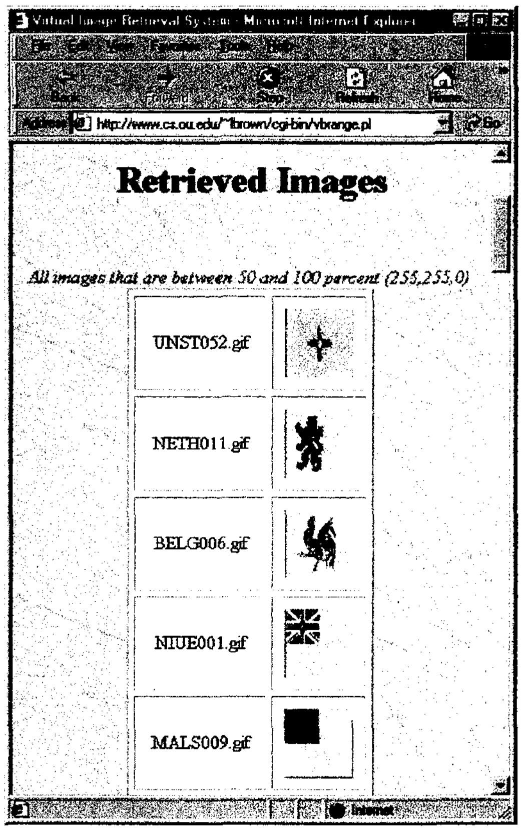
Retrieve Images by Color

Retrieve from the following database
- [✓ Flags of Countries (flags.net)]
- College Football Helmets (inside99.net)

all images where
no less than 50 %
and no more than 100 %

of the image contains the color
Red (0 - 255) 255
Green (0 - 255) 255
Blue (0 - 255) 0

Figure 8-2. User Interface for Submitting Range Queries to Prototype


Virtual Image Retrieval System - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.cs.ou.edu/~tbrown/cgi-bin/vbrange.pl
Go
Internet

Retrieved Images

All images that are between 50 and 100 percent (255,255,0)

UNST052.gif	
NETH011.gif	
BELG006.gif	
NIUE001.gif	
MALS009.gif	

Figure 8-3. User Interface for Displaying Retrieved Images

Since the prototype retrieves both virtual and binary images, the creation of the thumbnails differs. For binary images, the thumbnail is a version of the image reduced to 40x40 pixels. Since the prototype does not store an instantiated version of a virtual image, it cannot display a reduced version of that image, so the prototype displays a default picture for each virtual image it retrieves. When users click the thumbnail of a binary image, the system generates a web page containing the full size of the image. Alternatively, when users click the default picture corresponding to a virtual image, the system instantiates the image and displays a web page containing the derived image as displayed in Figure 8-4. An additional feature of the query results page displayed in

Figure 8-3 is that users can view the description of a virtual image by clicking on its filename as displayed in Figure 8-5

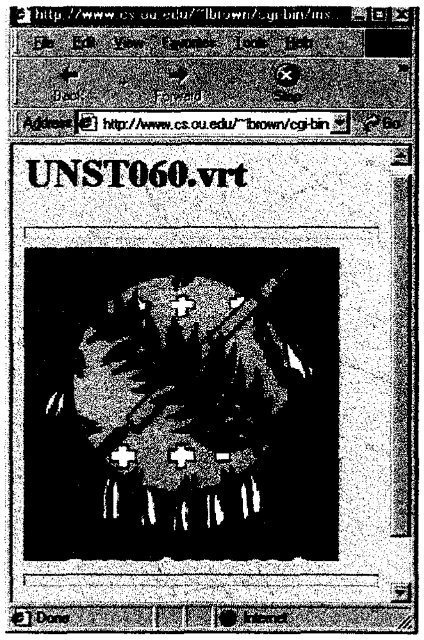
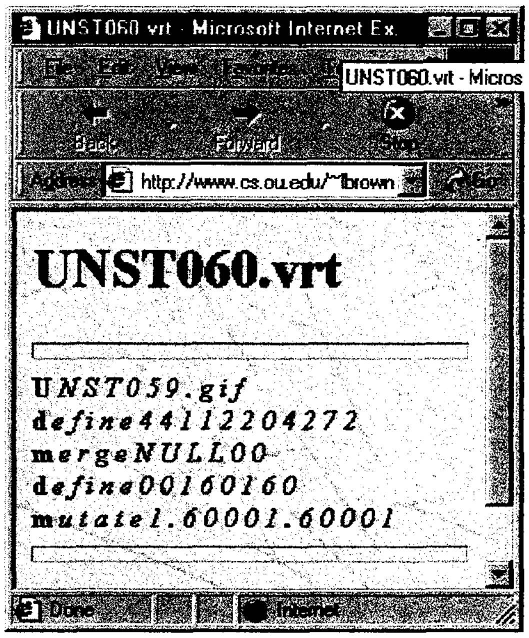


Figure 8-4. User Interface for Displaying an Instantiated Virtual Image


UNST060.vrt - Microsoft Internet Ex
http://www.cs.ou.edu/~lbrown
Back
Forward
Stop
Done
Internet

UNST060.vrt

UNST059.gif
define44112204272
mergeNULL00
define00160160
mutate1.60001.60001

Figure 8-5. User Interface for Displaying the Description of a Virtual Image

Users must specify a query image in order to submit a nearest neighbor query. In the VRS prototype, users can specify a query image using the browsing interface as displayed in Figures 8-6 and 8-7. In the first figure, the prototype allows users to specify the desired data set by selecting a random image displayed from each set. When users click on one of the images, the prototype randomly chooses 25 images and displays them as shown in the second figure. At this point, users may select any one of the 25 random images to be the query image by clicking its image. In order to select from a different set of images in the data set, users may regenerate the web page, which will again randomly choose 25 images.

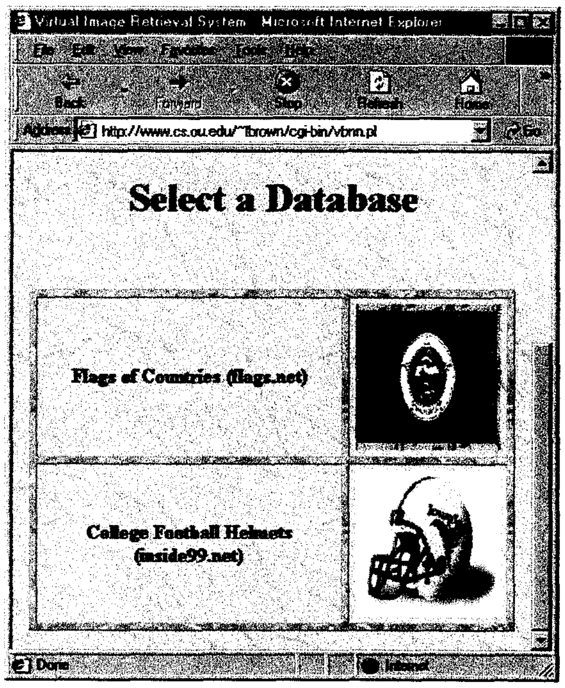

Virtual Image Retrieval System - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://www.cs.ou.edu/~fbrown/cgi-bin/vbnn.pl
Go
Select a Database
Flags of Countries (flags.net)
College Football Helmets (inside99.net)
Done
Internet

Figure 8-6. User Interface for Selecting a Data Set

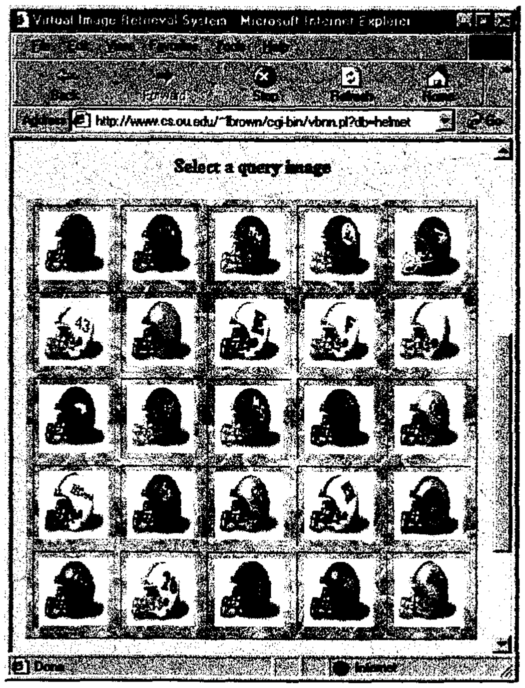

Virtual Image Retrieval System - Microsoft Internet Explorer
Back Forward Stop Refresh Home
Address http://www.cs.ou.edu/~fbrown/cgi-bin/vbnn.pl?db=helmet
Select a query image
Done
Internet

Figure 8-7. User Interface for Browsing Images of a Data Set

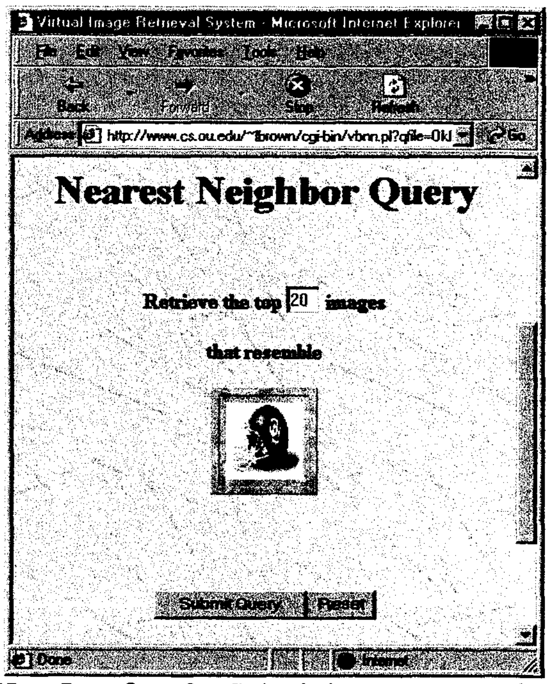

Virtual Image Retrieval System - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh
Address http://www.cs.ou.edu/~fbrown/cgi-bin/vbnn.pl?qfile=0kl
Go
Nearest Neighbor Query
Retrieve the top 20 images
that resemble
Submit Query
Reset
Done
Internet

Figure 8-8. User Interface for Submitting Nearest Neighbor Queries

When one of the 25 random images is clicked, the prototype uses it as the query image and generates the nearest neighbor query form that allows users to specify a value for k. An example of this form is displayed in Figure 8-8. After the query is submitted, the prototype executes the rule-based nearest neighbor retrieval algorithm and generates a web page containing the list of images that satisfy the query as shown previously in Figure 8-3.

8.2. Performance Parameters

The prototype developed in the previous section was used to compare and evaluate the performance of the four different image retrieval algorithms described earlier. The tests were conducted using three different sets of images collected from the Internet. The first set represents a set of college football helmets [Helm, 2003], and the second set represents a set of flags of countries [Flag, 2003]. Although these data sets represent real-world data, they have certain biases in them that may affect the test results, so a third image set was used in the performance evaluation. This final set was created as a collection of images representing various application areas including business, law enforcement, weather forecasting, and space exploration [Rand, 2003].

The default values for the variables used in the performance evaluations are listed in Tables 8-1 and 8-2. The variables N, N_{Binary}, $N_{Virtual}$, and N_{Op} will vary for each data set, so their default values will differ as well. This is also true for the variables N_{Main} and $N_{Unclass}$. Alternatively, the variables N_R, $N_{RBinary}$, $N_{RVirtual}$, and N_{Bounds} are dependent on the query posed to the prototype, so they are not given default values.

Param	Description	Default Value
N	Number of Images in the Database	551 (Helmet) 817 (Flag) 500 (Random)
N_{Binary}	Number of Binary Images in the Database	391 (Helmet) 466 (Flag) 5 (Random)
$N_{Virtual}$	Number of Virtual Images in the Database	160 (Helmet) 351 (Flag) 495 (Random)
N_R	Expected Number of Retrieved Images	Query-Dependent
$N_{Rbinary}$	Expected Number of Retrieved Images that are Binary	Query-Dependent
$N_{Rvirtual}$	Expected Number of Retrieved Images that are Virtual	Query-Dependent
N_{Op}	Average Number of Operations within a Virtual Image	4.56 (Helmet) 4.99 (Flag) 2.11 (Random)
N_{Bounds}	Expected No. of Times BOUNDS is executed for Nearest Neighbor Query	Query-Dependent
T_{Range}	Average Time needed to Identify Parameters of Range Query	0
T_{NN}	Average Time needed to Identify Parameters of Nearest Neighbor Query	0
T_{Access}	Average Time needed to Access an Image	0
$T_{Instant}$	Average Time needed to Instantiate a Virtual Image	$T_{Base} + T_{Access} + (N_{Op} \times T_{Op})$
$T_{Display}$	Average Time needed to Display an Image	0
$T_{Extract}$	Average Time needed to Extract a Histogram from a Binary Image	1.49
T_{Hist}	Average Time needed to Access and Compare Histogram Bins	0.000025
T_{Dist}	Average Time needed to Compute the Distance Between Histograms	0.0017
T_{Base}	Average Time needed to Identify the Base Image of a Virtual Image	0.00032
T_{Size}	Average Time needed to Identify the No. of Rows & Columns of an Image	0.00066
T_{Asize}	Average Time needed to Access the Numbers of Rows & Columns	0.000063
T_{Bounds}	Average Time needed to Execute the BOUNDS algorithm	$T_{Base} + T_{Access} + (N_{Op} \times T_{Rule})$
$T_{Virtual\_NN}$	Average Time needed to Execute the Virtual_NN algorithm	$N_{Bounds} \times T_{Bounds}$
T_{Rule}	Average Time needed to Apply a Rule for an Editing Operation	0.00028
T_{Op}	Average Time needed to Apply an Editing Operation to an Image	1.58
T_{Type}	Average Time needed to Determine if an Image is Binary or Virtual	0.00018
T_{AS}	Average Time needed to Add an Image ID to the Set of Satisfying Images	0
T_{AB}	Average Time needed to Add a Binary Image to the Database	0
T_{AV}	Average Time needed to Add a Virtual Image to the Database	0
T_{AH}	Average Time needed to Add a Histogram to the Database	0
S_{Binary}	Average Size of Binary Images in the Database (in bytes)	7281.88 (Helmet) 6204.28 (Flag) 11396.6 (Random)
$S_{Virtual}$	Average Size of Virtual Images in the Database (in bytes)	131.04 (Helmet) 137.16 (Flag) 70.27 (Random)
S_{Hist}	Average Size of Histogram Extracted From Binary Images (in bytes)	190.74 (Helmet) 228.27 (Flag) 190.40(Random)

Table 8-1. Default Values of Parameters Used in Performance Evaluation

Param	Description	Default Value
N_{Main}	Number of virtual images that contain only operations with bound-widening rules	14 (Helmet) 207 (Flag) 425 (Random)
$N_{Unclass}$	Number of virtual images that have an operation whose rule is not bound-widening	146 (Helmet) 144 (Flag) 70 (Random)
T_{Main}	Average Time needed to Access an Element of the Main Component	0.00005
$T_{Unclass}$	Average Time needed to Access an Element of the Unclassified Component	0
$T_{AddMain}$	Average Time needed to Add an Element to the Main Component	0.0025
$T_{AddUnclass}$	Average Time needed to Add an Element to the Unclassified Component	0.0010
S_{ID}	Average Size of an Identifier (in bytes)	18.21

Table 8-2. Default Values of Data Structure Parameters used in Performance Evaluation

All of the Time (T) variables are expressed in seconds. Some of the expected times are directly dependent on a data set. For those variables, a default time is given for each of the flag and helmet data sets. The default values for each of the Time variables were determined by executing the prototype for each action. In addition, the Size (S) variables are all dependent upon the data sets. Thus, a default value, which is expressed in bytes, is given for each of the three data sets.

Table 8-3 describes the dynamic parameters used in the evaluation. The percentage of virtual images was varied in the tests that evaluated the execution time and permanent storage space of the algorithms. The query parameters were varied in the tests that evaluated the retrieval accuracy.

Param	Parameter	Range of Values	Default Value
$P_{Virtual}$	Percentage of Images in the Database Stored Virtually ($N_{virtual}/N$)	0-29% (Helmet) 0-43% (Flag) 0-95% (Random)	29% (Helmet) 43% (Flag) 99% (Random)
Width	Range specified by parameters of range query (PCT_{max}-PCT_{min})	5 - 75	25
k	Number of images requested by parameters of k-nearest neighbor query	5 - 100	25

Table 8-3. Dynamic Parameters used in Performance Evaluation

8.3. Performance Evaluation Results

This section presents the results of the various sets of tests performed to measure the accuracy, execution time, and permanent storage space of the four different approaches to image retrieval: Rule-based, BSH, VSIS, and VSII. The first set of tests examines how the permanent storage space is affected by varying the percentage of images in the database that are stored virtually. This test will illustrate one of the main advantages of using virtual images over the BSH approach. The second set of tests examines the effect of the same parameter on the time needed to execute retrieval queries. The third set of tests examines the effect of the percentage of virtual images in the database on the time needed to insert edited images. The final set of tests examines how the parameters of each query affect the retrieval accuracy of the rule-based algorithms.

8.3.1. Permanent Storage Space

The first test measured the permanent storage space required by the conventional BSH approach and the approaches that utilize virtual images while varying the percentage of images stored virtually and keeping the total number of images constant. The results of the test for the helmet, flag, and random data sets are displayed in Figures 8-9a, 8-9b, and 8-9c, respectively. Each figure indicates that the virtual approaches consume much less space than the BSH approach.

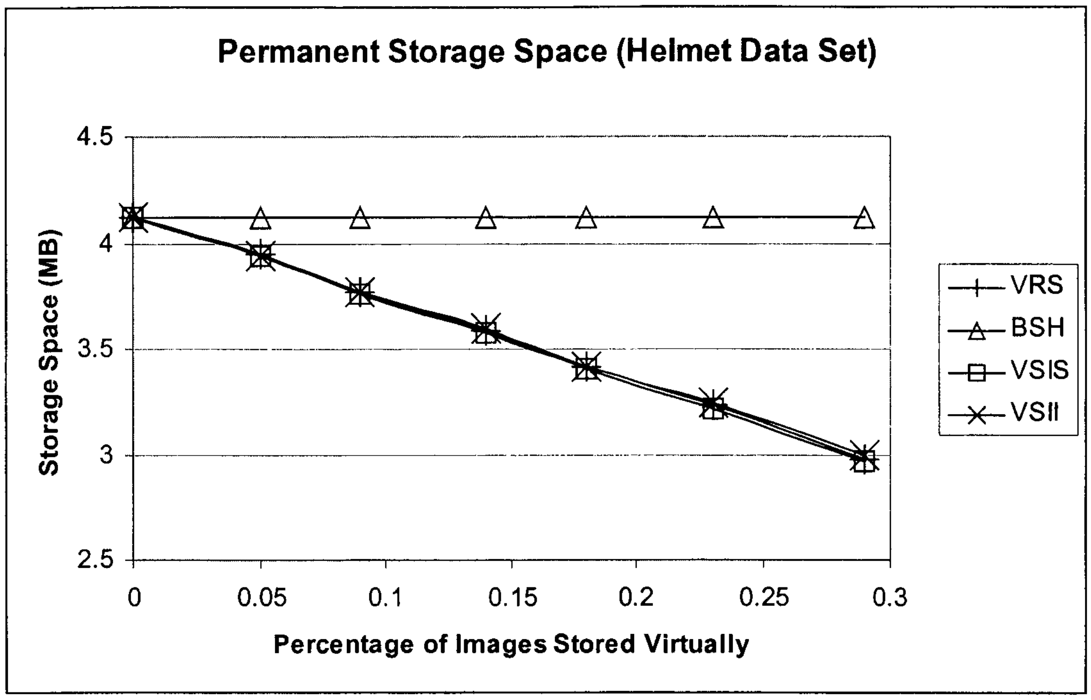


Figure 8-9a. Space Savings vs. Percentage of Images Stored Virtually (Helmet Data Set)

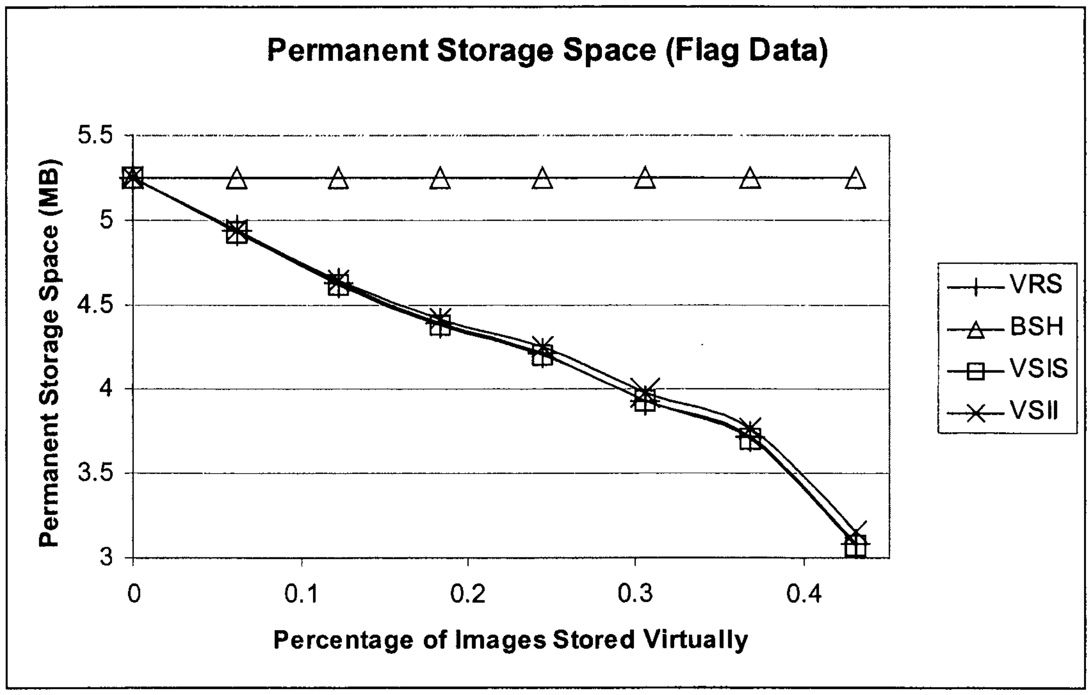


Figure 8-9b. Space Savings vs. Percentage of Images Stored Virtually (Flag Data Set)

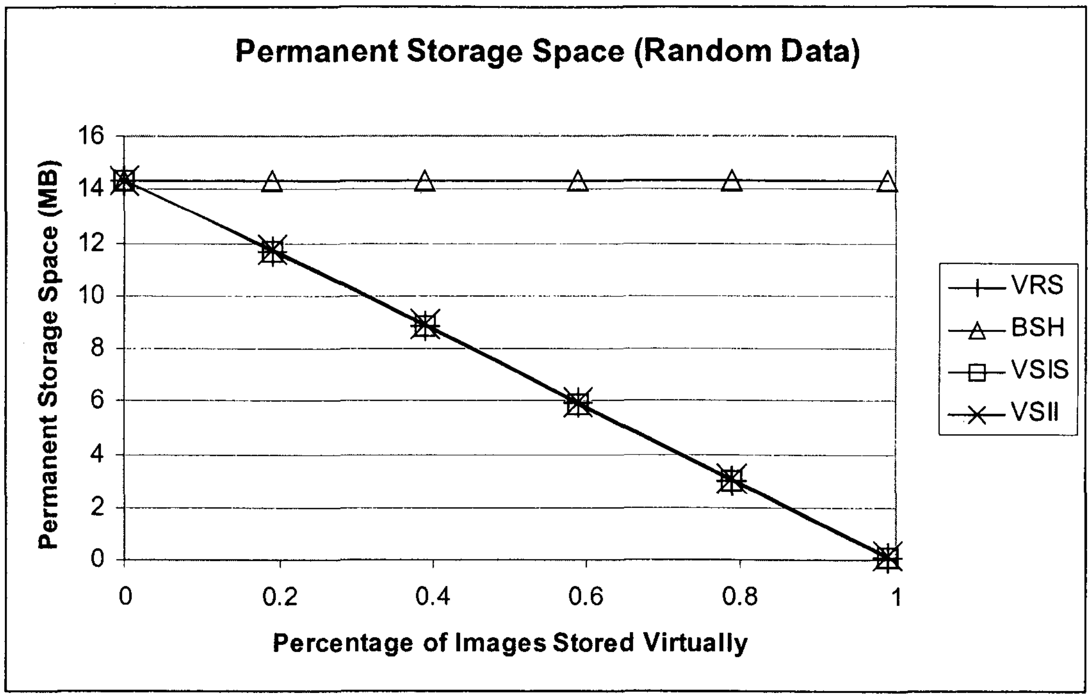


Figure 8-9c. Space Savings vs. Percentage of Images Stored Virtually (Random Data Set)

Because the binary images compose the largest portion of the total space used by the approaches, the difference in space is directly related to the percentage of images that are stored virtually. For the helmet data set, the virtual approaches store 29% fewer binary images than the BSH approach, and they use 28% less space. Similarly, for the flag data set, the virtual approaches store 43% fewer binary images than the BSH approach, and they use 41% less space. Finally, the virtual approaches store 99% fewer images than the BSH approach for the random data set, and they use 99% less space. Since the percentage of images stored virtually directly affects the space saved, it can be concluded that applications that need to reduce the amount of space consumed by data will benefit by storing as many images as possible virtually. Such applications include those that archive data or transmit data across a network.

8.3.2. Retrieval Time

The next set of tests compared the time required to process retrieval queries using the algorithms for the various approaches. To determine the average time used by the VRS prototype to process range queries, 27 queries were executed with random values for the query parameters PCT_{min}, PCT_{max}, and C_Q for each data set. This number of executions permitted computing the mean average time with a relative error below 0.1. Similarly, 27 queries were executed with random values for the query parameters Q and k to determine the average time used to process nearest neighbor queries.

Figures 8-10a, 8-10b, and 8-10c show the results for the helmet, flag, and random data sets, respectively. These results compare the time taken by the VSIS and rule-based approaches for processing nearest neighbor queries against the percentage of images stored virtually. Each of the illustrates demonstrates the main disadvantage of the VSIS approach, namely that it requires much more time than the other methods to process the retrieval queries. For example, the rule-based algorithm is an average of 99.40% faster than the VSIS approach for the helmet data set, an average of 99.87% faster than the VSIS approach for the flag data set, and an average of 99.95% faster than the VSIS approach for the random data set.

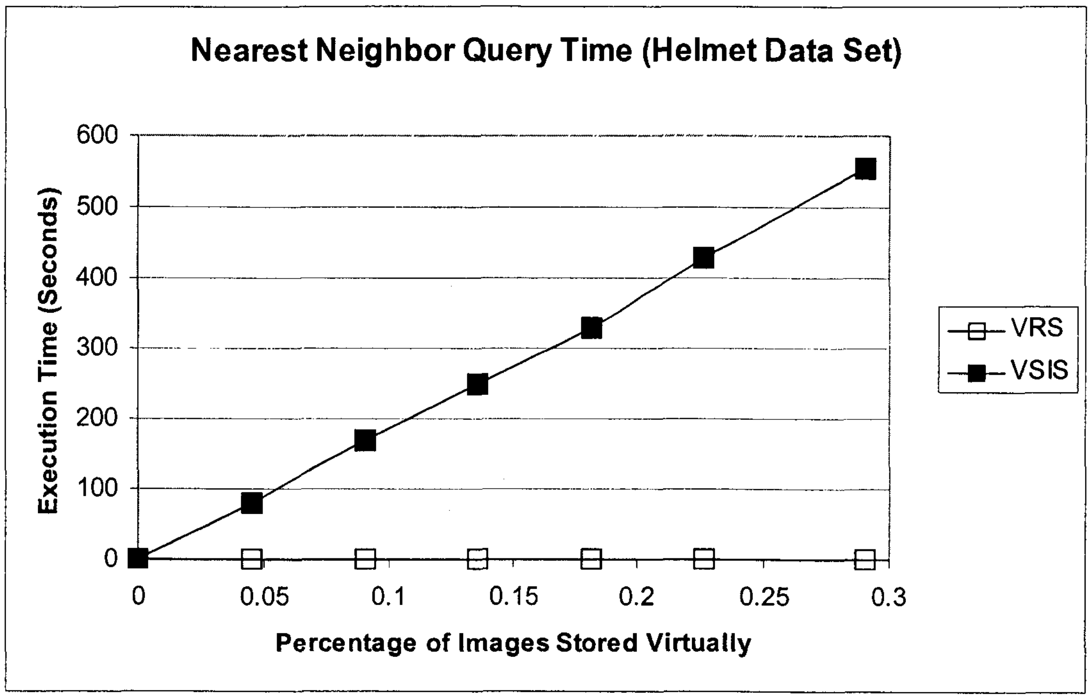


Figure 8-10a. Searching Time for Nearest Neighbor Query vs. Percentage of Images Stored Virtually (Helmets)

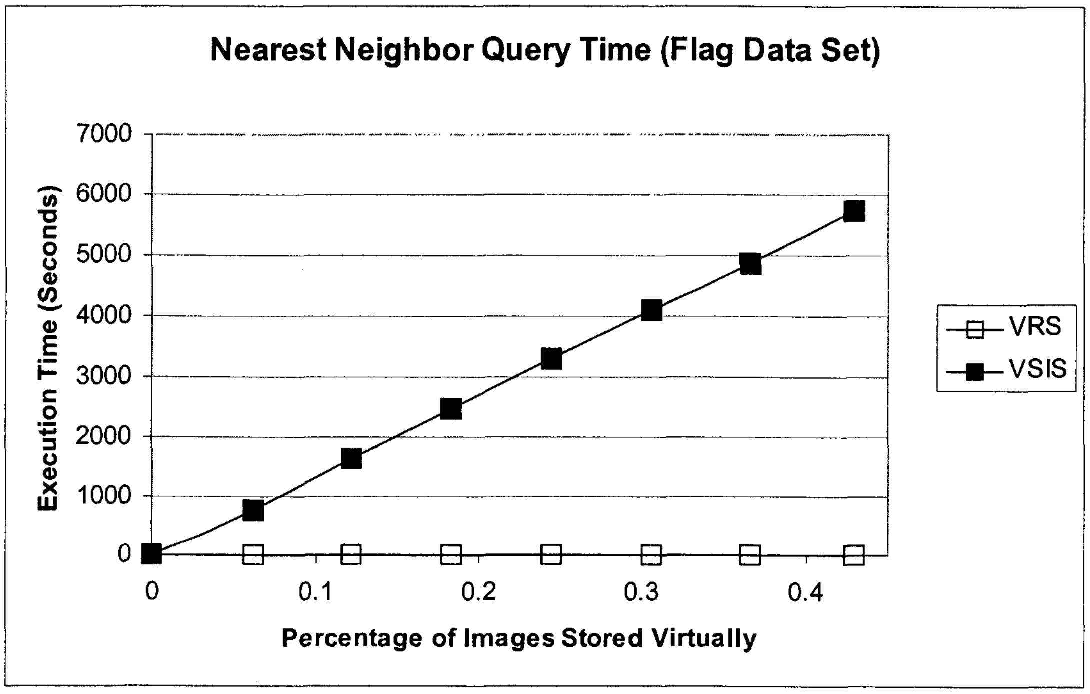


Figure 8-10b. Searching Time for Nearest Neighbor Query vs. Percentage of Images Stored Virtually (Flags)

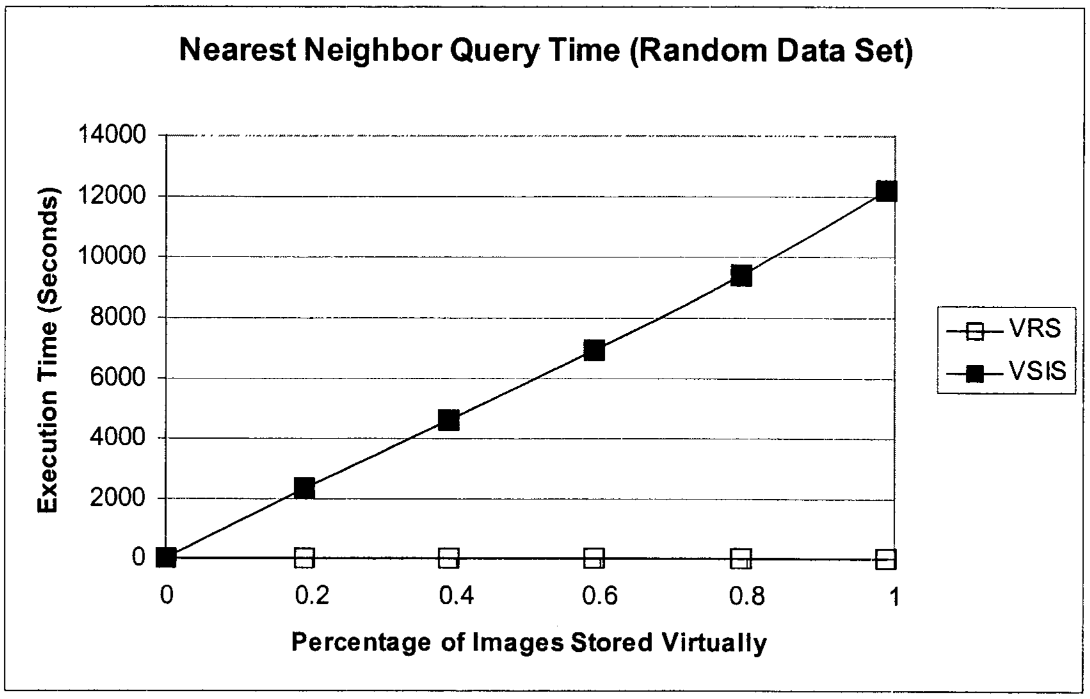


Figure 8-10c. Searching Time for Nearest Neighbor Query vs. Percentage of Images Stored Virtually (Random)

Similar results are obtained when the execution time for the rule-based and VSIS approaches are presented for the range query. These results are in Figures 8-11a, 8-11b, and 8-11c for the helmet, flag, and random data set, respectively. During the testing, the rule-based approach was an average of 99.39% faster for the helmet data set, 99.99% faster for the flag data set, and an average of 99.99% faster for the random data set when compared to the VSIS approach.

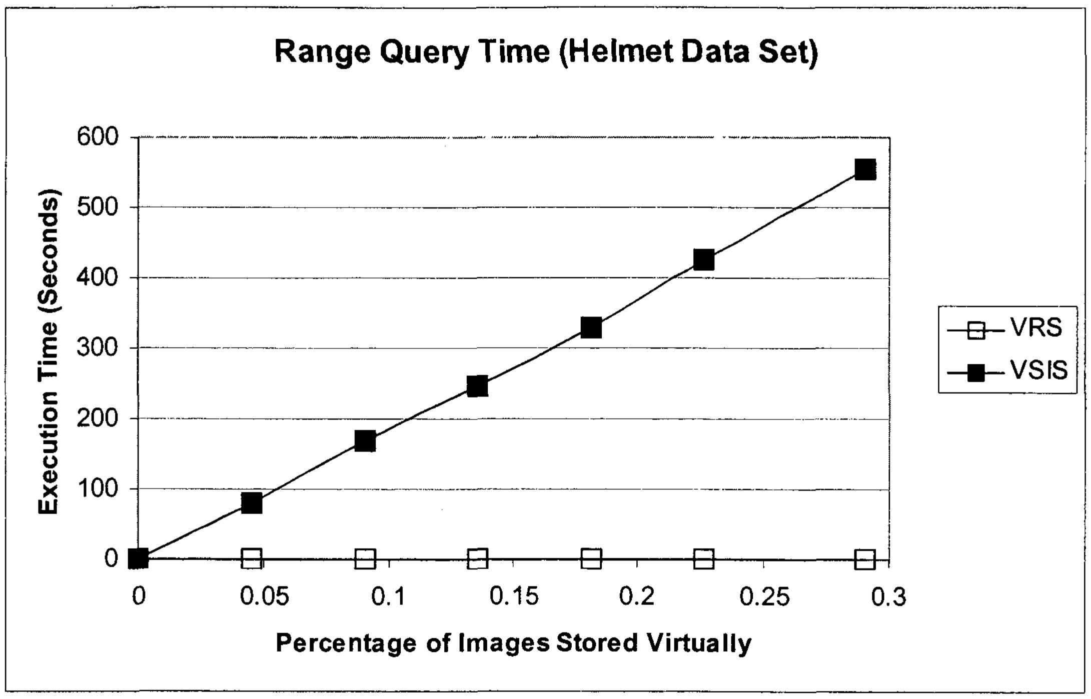


Figure 8-11a. Searching Time for Range Query vs. Percentage of Images Stored Virtually (Helmets)

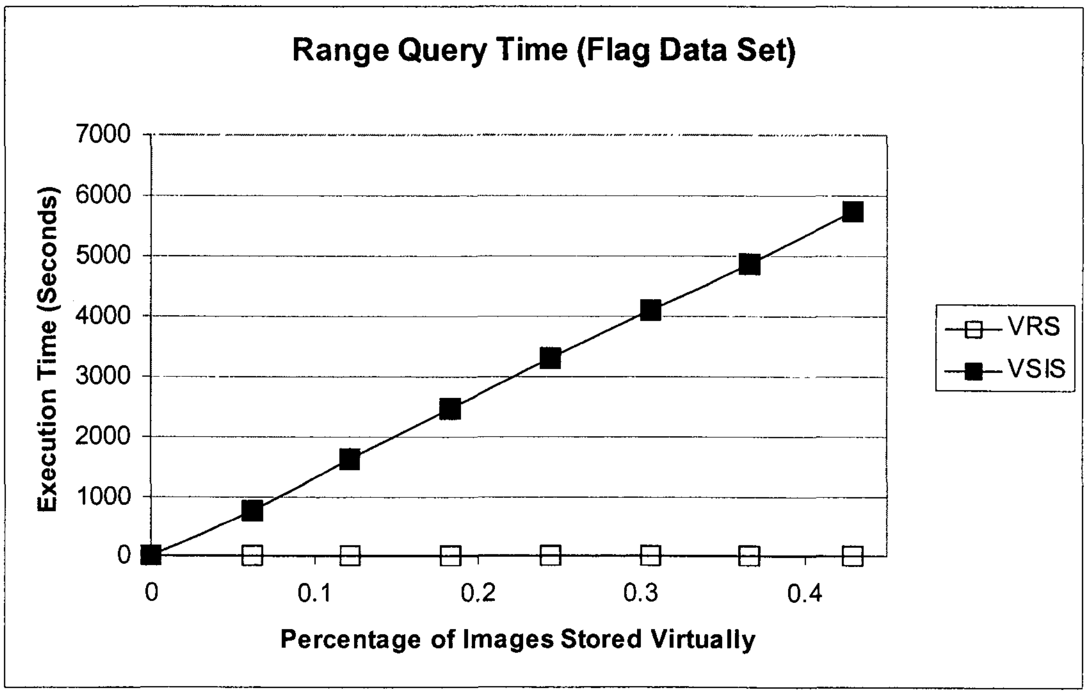


Figure 8-11b. Searching Time for Range Query vs. Percentage of Images Stored Virtually (Flags)

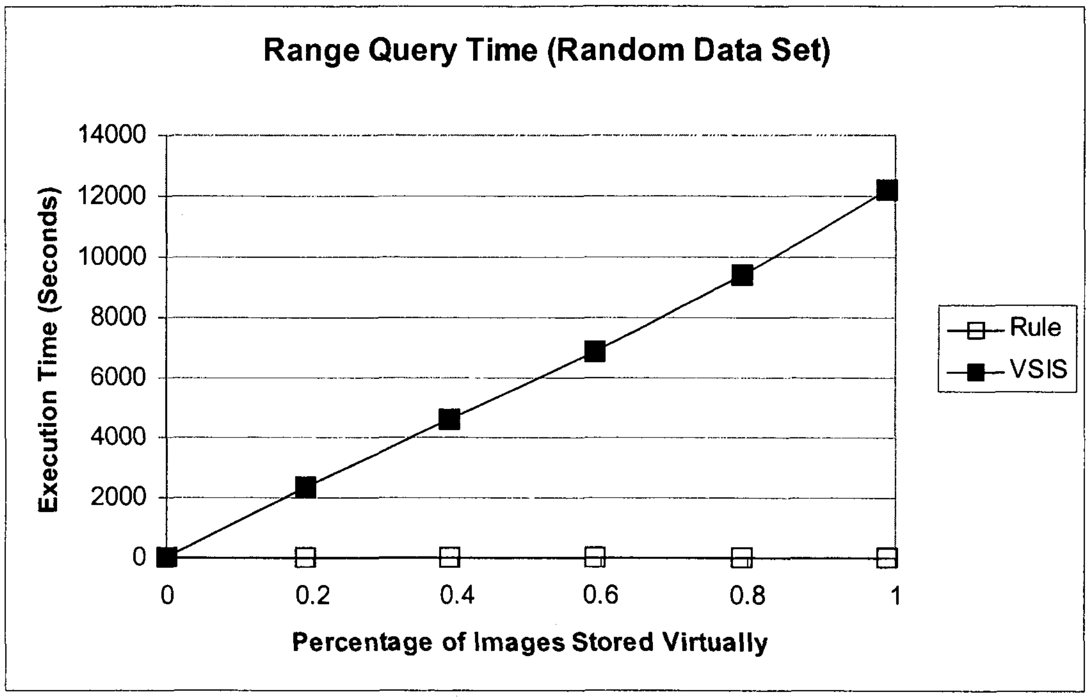


Figure 8-11c. Searching Time for Range Query vs. Percentage of Images Stored Virtually (Random)

The results of all of the above figures indicate that the VSIS approach behaves worse as more images are stored virtually unlike the rule-based approach. The reason for this behavior is that the each virtual image is instantiated as a part of the VSIS approach. Consequently, an increase in the percentage of images stored virtually means an increase in the number of instantiations performed during query processing.

Tests were also conducted to compare the time required by the rule-based algorithm for processing retrieval queries to the time required by the BSH and VSII approaches which are histogram-based. The first test measured the average execution times of the different approaches for processing nearest neighbor queries. The results of the test are displayed in Figure 8-12a, 8-12b, and 8-12c for the helmet, flag, and random data sets, respectively. They indicate that the rule-based algorithms get slower as the percentage of virtual images increases. In contrast, the time of the histogram-based algorithms remain constant since they search N feature vectors irrespective of the percentage of images that are stored virtually. Thus, the rule-based algorithm does execute slower than the histogram-based ones. On average, the histogram-based algorithms were 21.04% faster for the helmet data set, 16.72% faster for the flag data set, and 14.06% faster for the random data set.

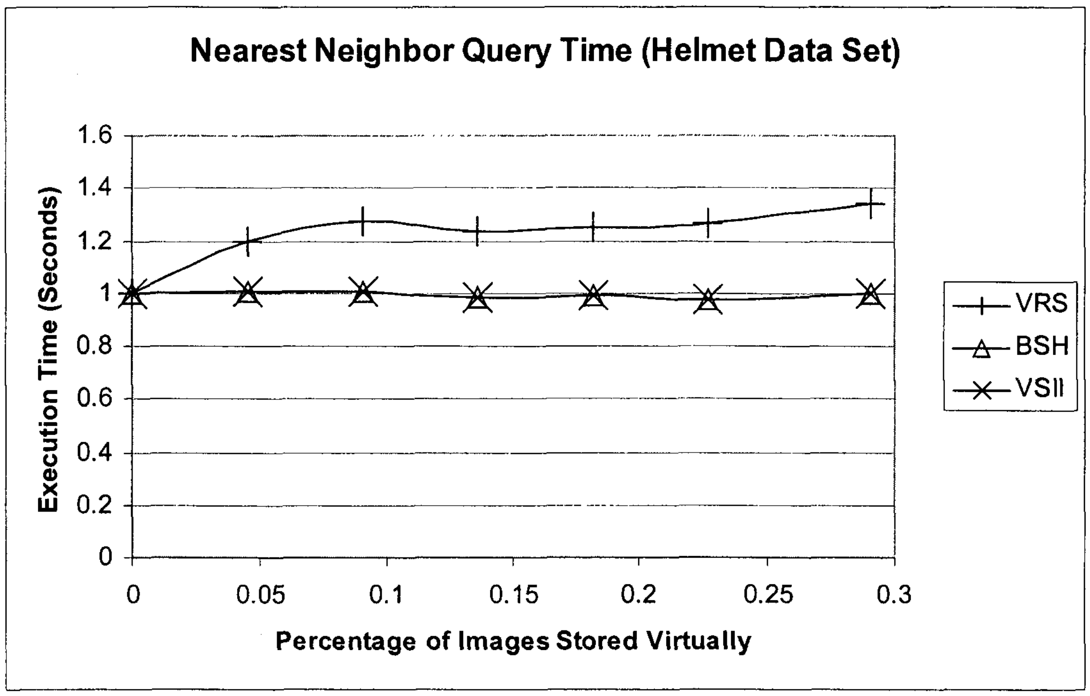


Figure 8-12a. Searching Time for Nearest Neighbor Query vs. Percentage of Images Stored Virtually (Helmets)

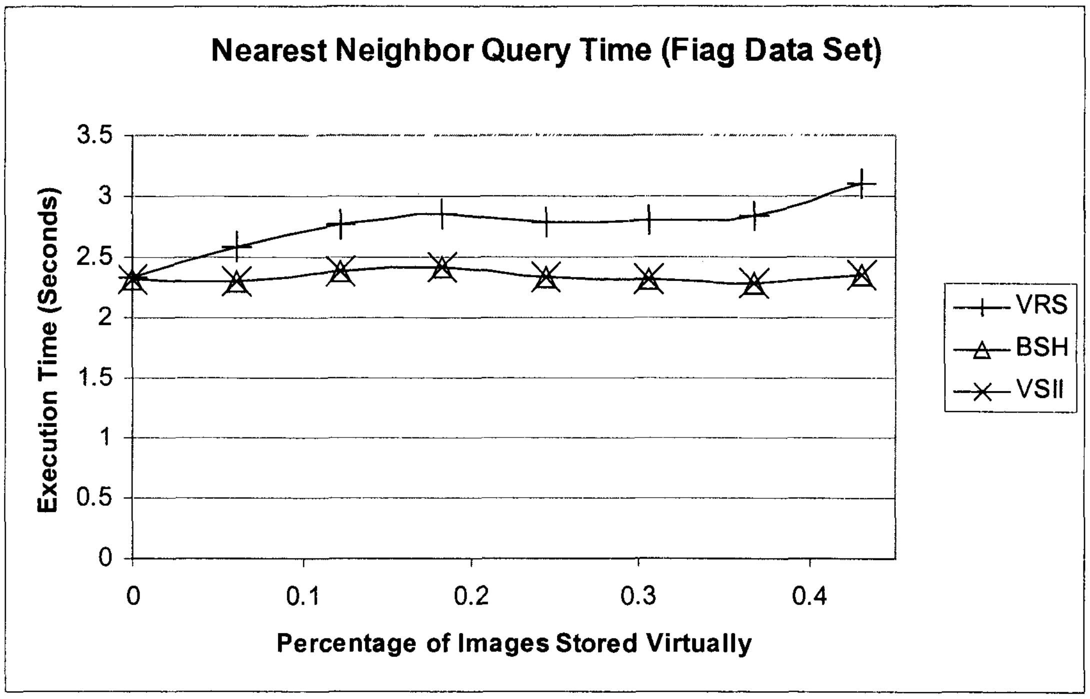


Figure 8-12b. Searching Time for Nearest Neighbor Query vs. Percentage of Images Stored Virtually (Flags)

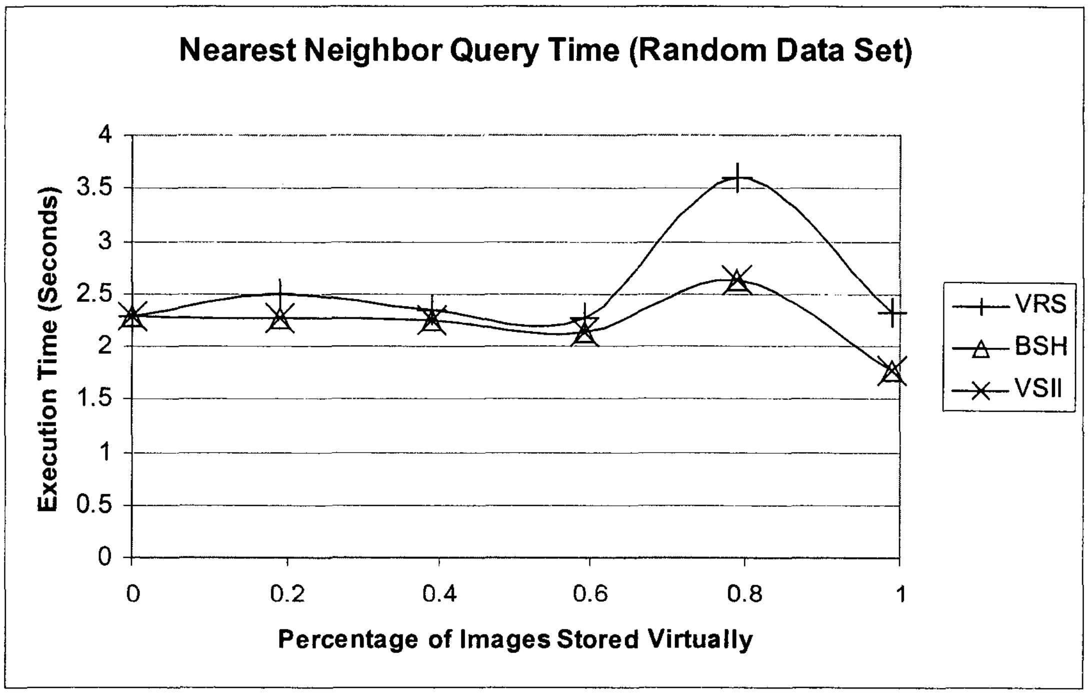


Figure 8-12c. Searching Time for Nearest Neighbor Query vs. Percentage of Images Stored Virtually (Random)

The next test indicates one of the causes of the slower execution time of the rule-based approach by measuring the average time needed to process range queries. The results of this test are displayed in Figures 8-13a, 8-13b, and 8-13c, respectively, and they demonstrate that the rule-based algorithm requires substantially more time to execute than the histogram-based algorithms. The reason for the increase in time is that the rule-based algorithm applies a rule for each editing operation in a virtual image to determine the colors in an image while the other approaches simply access a single histogram value. The rule-based algorithm for processing k-nearest neighbor queries uses this range query algorithm repeatedly which means that if the time needed by this algorithm could be reduced, it would reduce the time needed by the rule-based k-nearest neighbor query processing algorithm.

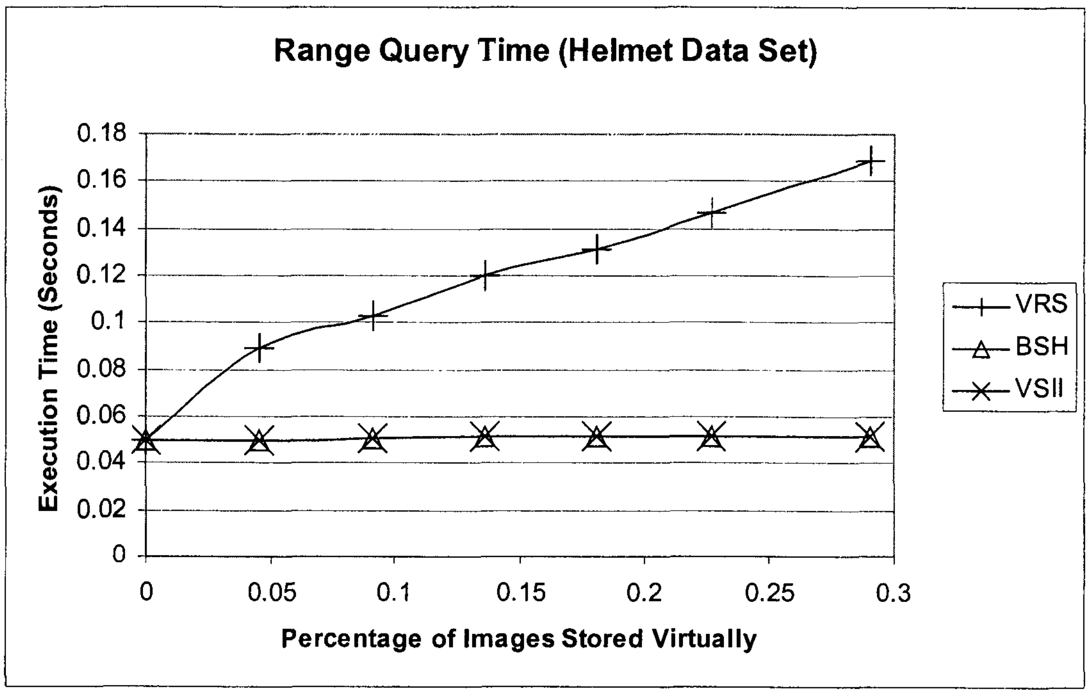


Figure 8-13a. Searching Time for Range Query vs. Percentage of Images Stored Virtually (Helmets)

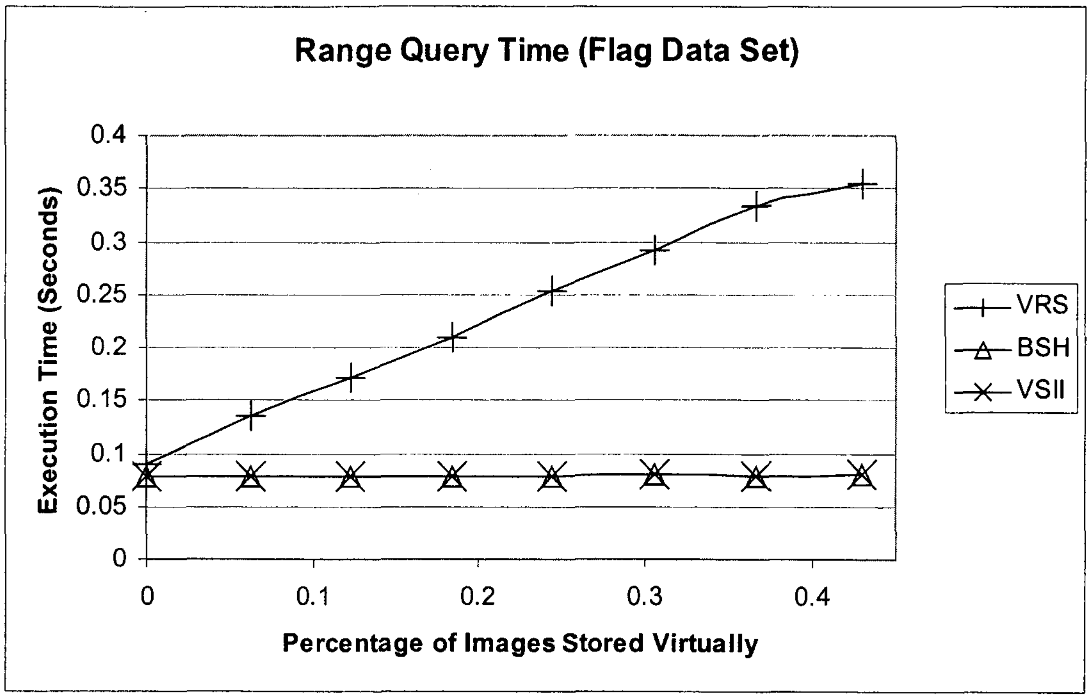


Figure 8-13b. Searching Time for Range Query vs. Percentage of Images Stored Virtually (Flags)

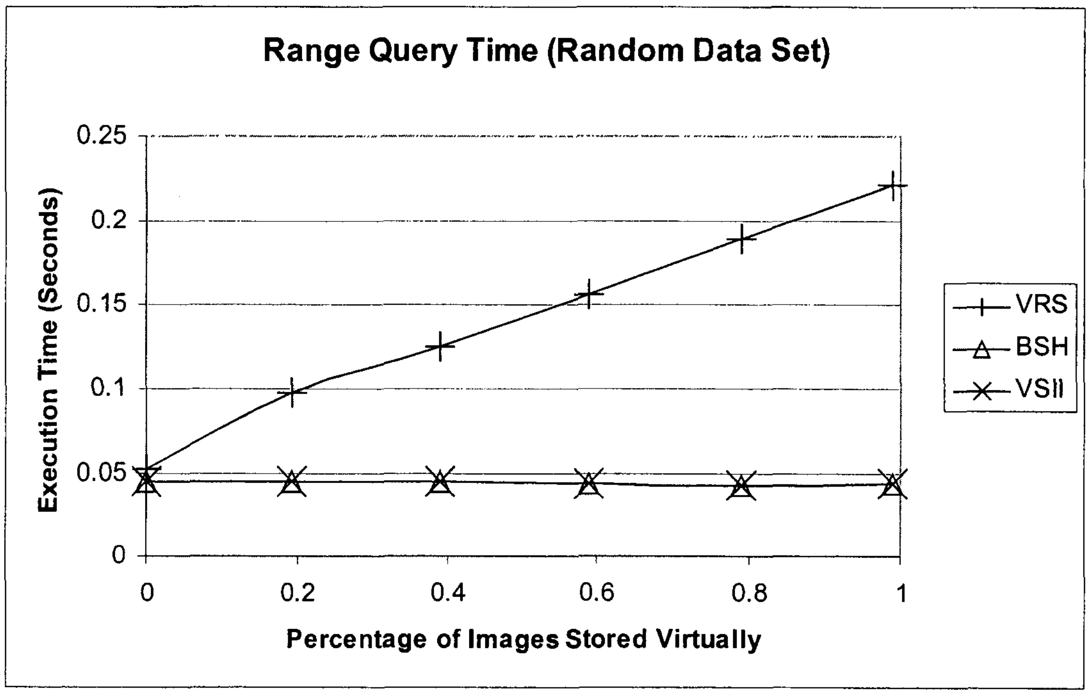


Figure 8-13c. Searching Time for Range Query vs. Percentage of Images Stored Virtually (Random)

Speeding up the rule-based query processing algorithm is one of the goals of the data structure proposed in Chapter 5. The next test compared the average execution time of the rule-based algorithm for processing range queries with and without that data structure. As with the previous tests, the average execution time is measured against the percentage of the images in the database that are stored virtually.

The results of the above test are displayed in Figures 8-14a, 8-14b, and 8-14c for the flag, helmet, and random data sets, respectively. They indicate that the average execution time of the proposed data structure is smaller than the average execution time without it. Specifically, using the proposed data structure allows the rule-based approach to process the range queries an average of 33.07% faster for the helmet data set, an average of 22.08% faster for the flag data set, and an average of 18.03% faster for the random data set. The reason for the improvement is that the data structure processes the queries while avoiding the application of some of the rules for virtual images.

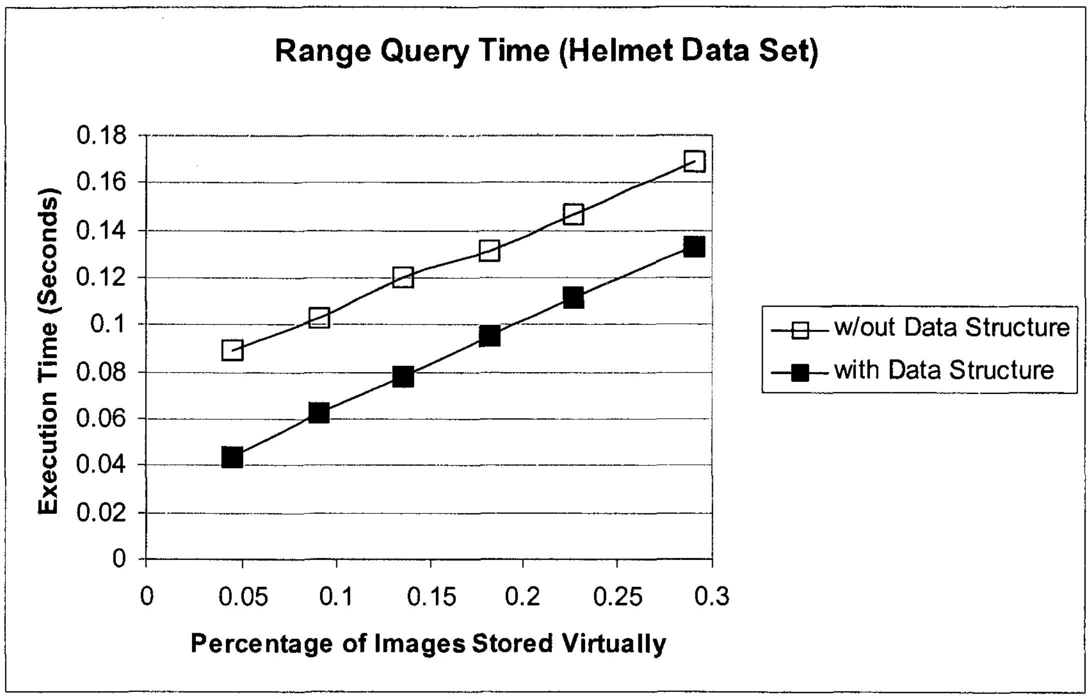


Figure 8-14a. Searching Time for Range Query vs. Percentage of Images Stored Virtually (Helmets)

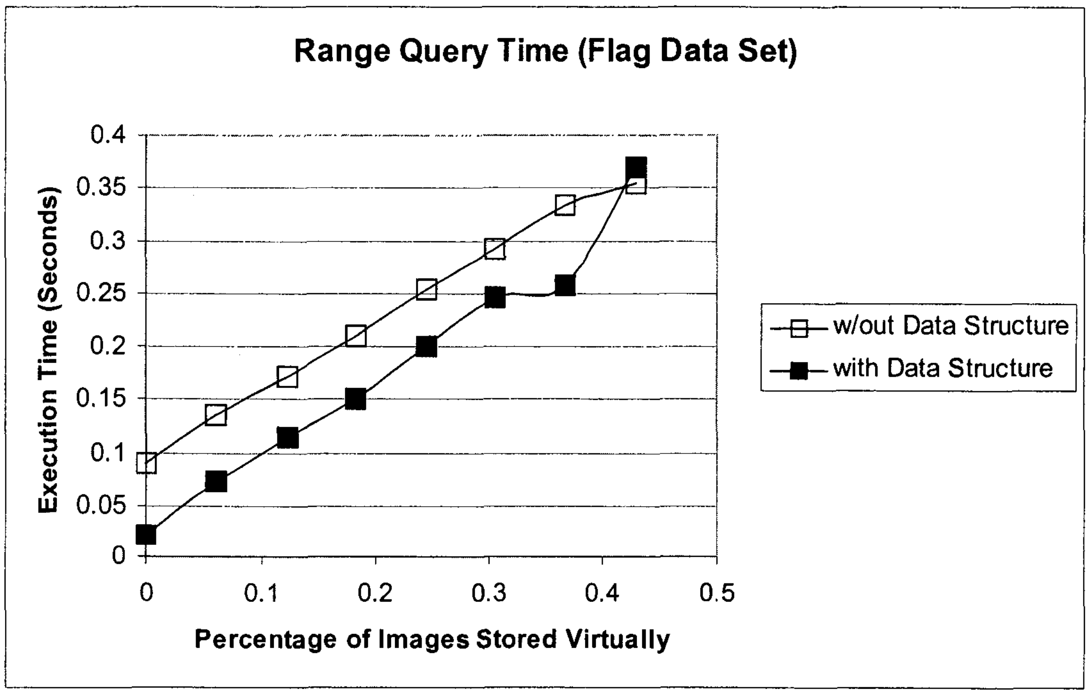


Figure 8-14b. Searching Time for Range Query vs. Percentage of Images Stored Virtually (Flags)

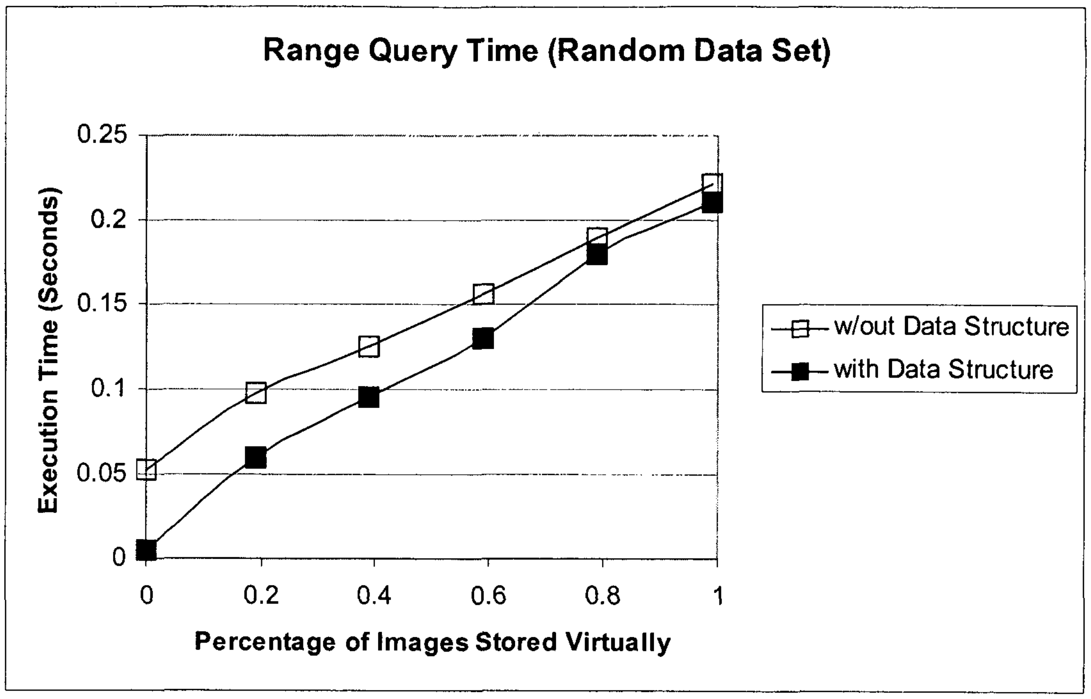


Figure 8-14c. Searching Time for Range Query vs. Percentage of Images Stored Virtually (Random)

8.3.3. Insertion Time

The next set of tests compared the time required to insert images using the various approaches for retrieving images. The time calculated for inserting images into the database was determined as the time needed to extract the features from the images stored in a conventional binary format. Figures 8-15a, 8-15b, and 8-15c show the results of comparing the insertion time of the rule-based and VSII approaches for the helmet, flag, and random data sets, respectively. Each of the figures demonstrates the main disadvantage of the VSII approach, namely that it requires much more time than the other approaches since it instantiates virtual images to extract their features. For example, the rule-based algorithm is an average of 51.56% faster for the helmet data set, an average of 71.59% faster for the flag data set, and an average of 92.28% faster for the random data set over the VSII approach.

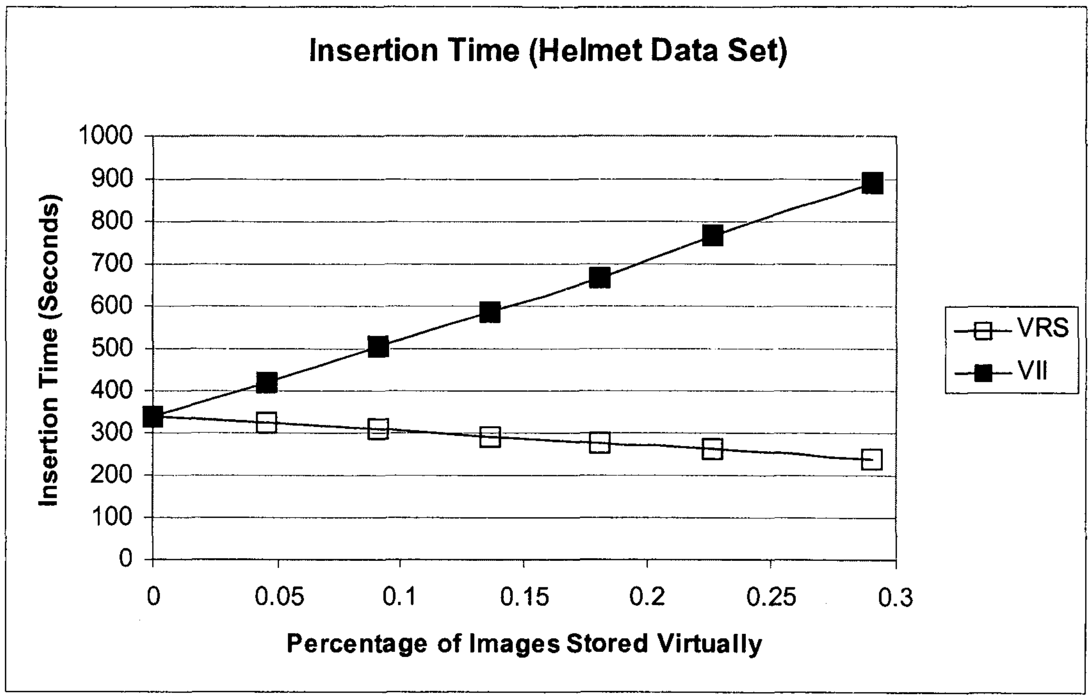


Figure 8-15a. Insertion Time vs. Percentage of Images Stored Virtually (Helmets)

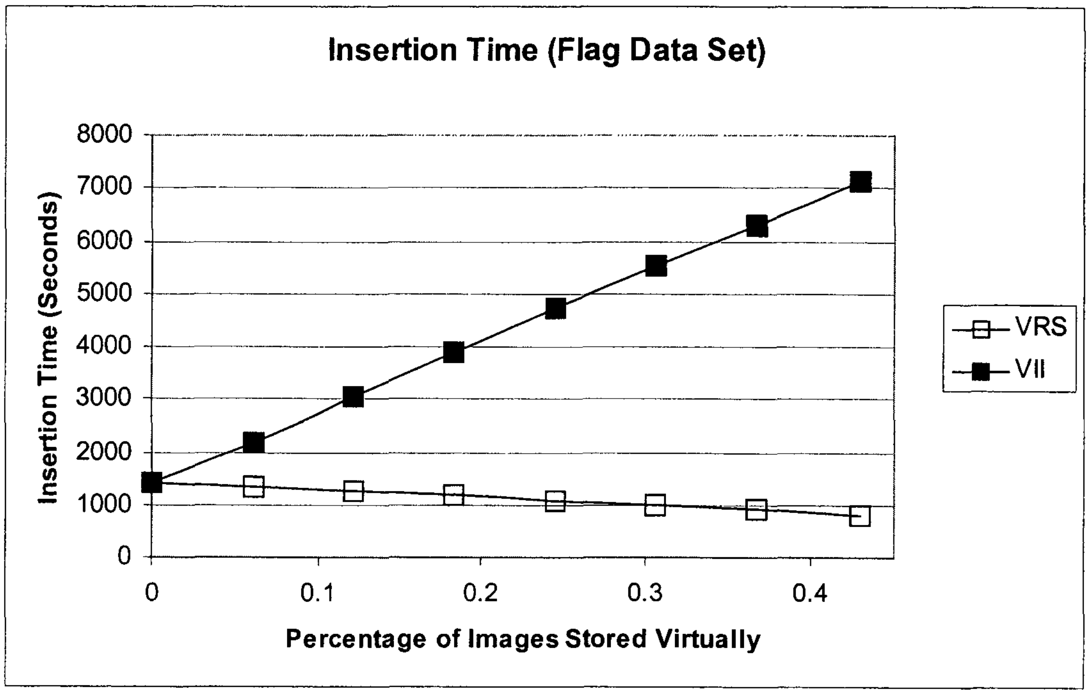


Figure 8-15b. Insertion Time vs. Percentage of Images Stored Virtually (Flags)

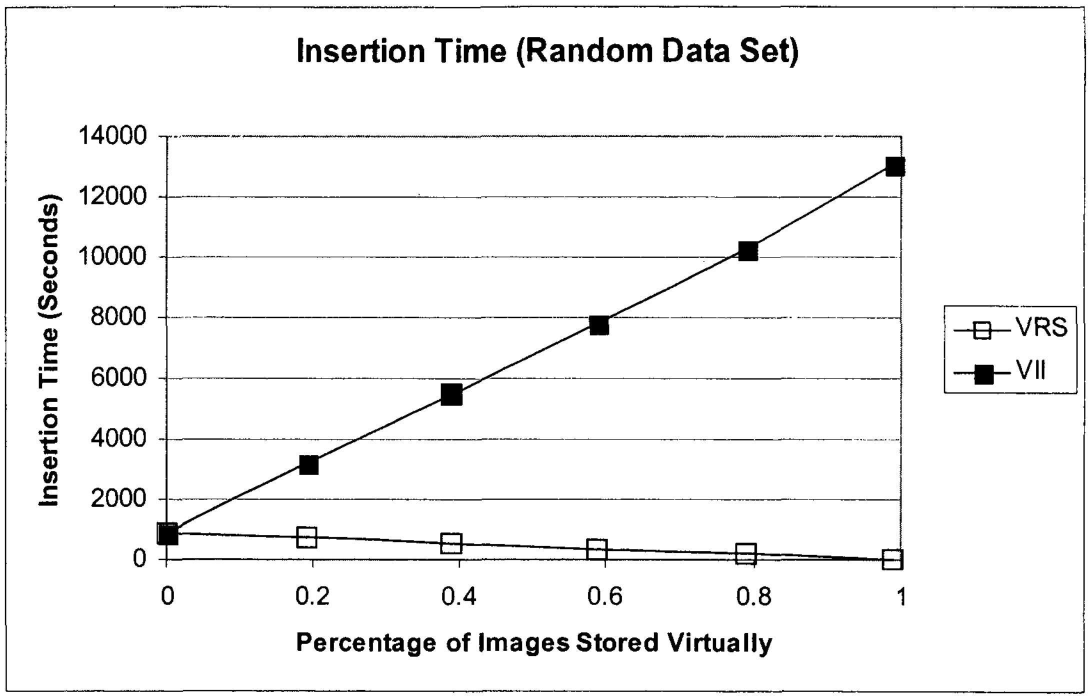


Figure 8-15c. Insertion Time vs. Percentage of Images Stored Virtually (Random)

When comparing the times required to insert images using the BSH approach and the rule-based and VSIS approaches, it is sufficient to note that the insertion time is computed as the time needed to extract histograms from the binary images. BSH stores all images in a binary format while the rule-based and VSIS approaches store only those that are not edited. Thus, BSH stores more binary images than the rule-based and VSIS, which means that it must extract more histograms. Consequently, it can be determined that BSH will take longer than the rule-based and VSIS approaches without testing the prototypes. The difference in the times will be directly proportional to the number of images in the rule-based and VSIS approaches that are not stored in a binary format.

8.3.4. Retrieval Accuracy

The results of the tests in the previous sections indicate that the rule-based algorithms use much less permanent storage space than the BSH algorithms, require much less retrieval time than the VSIS algorithms, and require much less insertion time than the VSII algorithms. Thus, the rule-based approach is the only one among the four that does not perform poorly in at least one of the three areas, permanent storage space, retrieval query processing time, and insertion query processing time. The VSII and VSIS approaches, however, have an advantage over the rule-based approaches in that they use histograms to compare and retrieve images. Thus, they will produce the same results in response to retrieval queries as the BSH approach. The purpose of the tests in this section is to illustrate the difference between the results of the rule-based approach and the histogram-based approaches.

The metrics used to determine the accuracy of the rule-based algorithms in the following set of tests were precision and recall. Precision is the number of relevant images retrieved divided by the total number of images retrieved, and recall is the number of relevant images retrieved divided by the total number of relevant images in the database [Falo, 1996]. Since the k-nearest neighbor specifies the number of images obtained from the database by the retrieval algorithms, both the precision and the recall of the algorithms will be the same. To illustrate, consider if an algorithm retrieves x of the k relevant images, where x is some number less than or equal to k. The result is that the recall of the algorithm would be x/k. In addition, the algorithm returned k images as well, which means that only x of those k images are relevant. Thus, the precision of the algorithm would also be x/k.

8.3.4.1. Accuracy of Rule-Based Nearest Neighbor Query Processing Algorithm

This set of tests compared the precision and recall of the various approaches for processing nearest neighbor queries of the type "*Retrieve the k images that are the most similar to the query image Q*", where k is an integer and Q is a image stored in a binary format. The number of images that are retrieved should affect the accuracy of the rule-based algorithm, so the first test measured the precision and recall against the number of images retrieved by the algorithm, which is k. The values of k ranged from 5 to 50 for the flag and helmet data set, and from 5 to 100 for the random data set.

The results of this test are displayed in Figures 8-16a, 8-16b, and 8-16c for the helmet, flag, and random data sets. The results indicate that the histogram-based algorithms perform slightly better than the rule-based algorithm in terms of retrieval accuracy. Specifically, the rule-based algorithm retrieved 12.36% fewer relevant images than the histogram approaches for the helmet data set, 25.57% fewer relevant images for the flag data set, and 11.22% fewer relevant images for the random data set.

The above results also indicate that the rule-based algorithm performs better when users search for many images that are similar to a query image than when users search for the a smaller number of images. Thus, the rule-based algorithm is more appropriate when a user can pose a query requesting many images of flags that are similar in color to the U.S. flag instead of requesting the image of the flag that is most similar to it. An example application that requires retrieving many images is one for law enforcement where a query may be posed that requests pictures of all suspects that are similar to a drawing formed by a sketch artist. Because of the uncertainty in the drawing, the system should retrieve many images of suspects that may be similar.

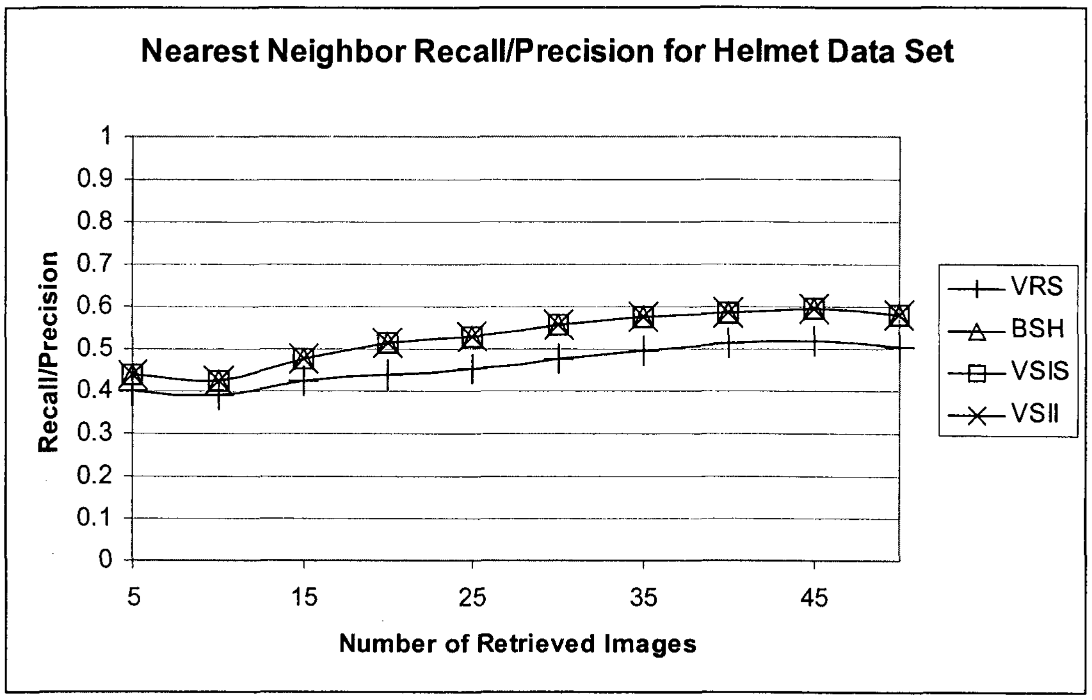


Figure 8-16a. Retrieval Accuracy vs. Number of Retrieved Images (Helmets)

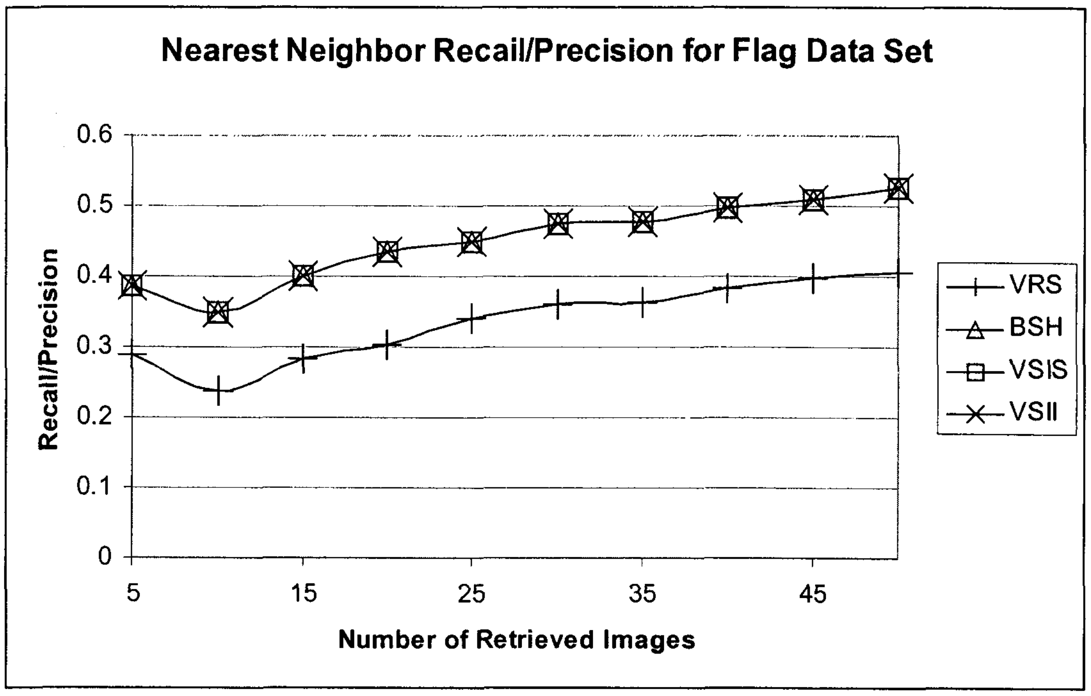


Figure 8-16b. Retrieval Accuracy vs. Number of Retrieved Images (Flags)

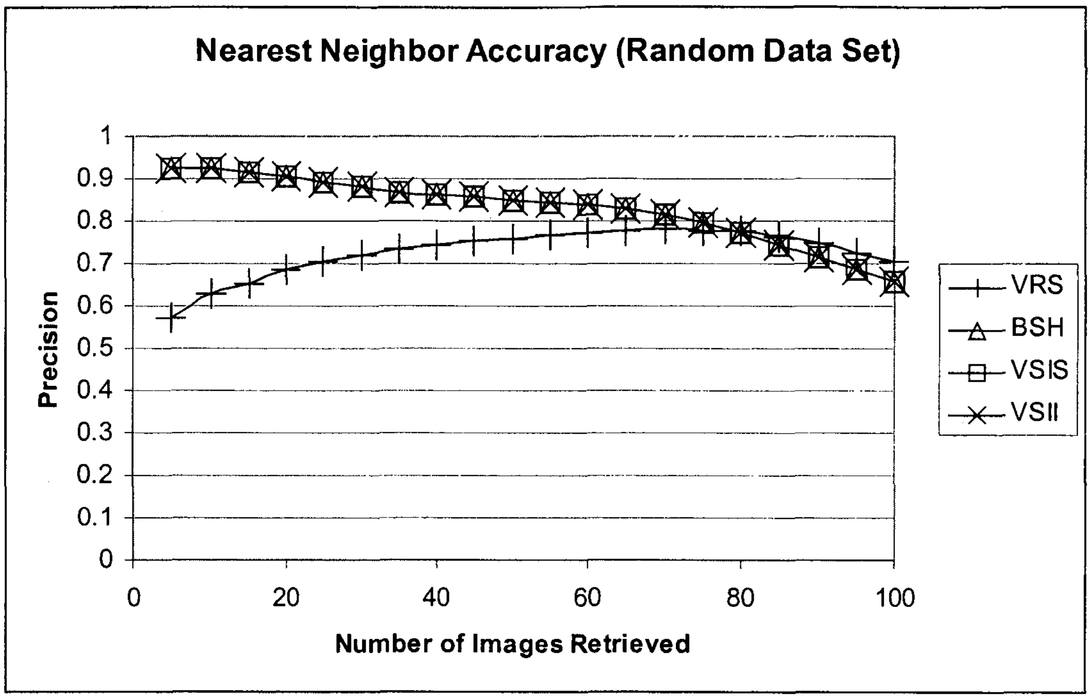


Figure 8-16c. Retrieval Accuracy vs. Number of Retrieved Images (Random)

8.3.4.2. Accuracy of Rule-Based Range Query Processing Algorithm

One of the factors affecting the retrieval accuracy of the rule-based algorithm for processing k-nearest neighbor queries is the retrieval accuracy of the range query processing algorithm. The next set of tests measured that accuracy in order to identify the cause of the results of the previous test. Since the rule-based range query processing algorithm works by computing bounds on the percentage of pixels that may be of the desired query color, and it retrieves an image if those bounds intersect the range [PCT_{min}, PCT_{max}], then the range specified in the query will affect whether the algorithm retrieves an image. Thus, the query range affects the precision and recall of the rule-based algorithm. Consequently, the next test measures the precision and recall against the width of the query range [PCT_{min}, PCT_{max}] as it increased from 5 to 75

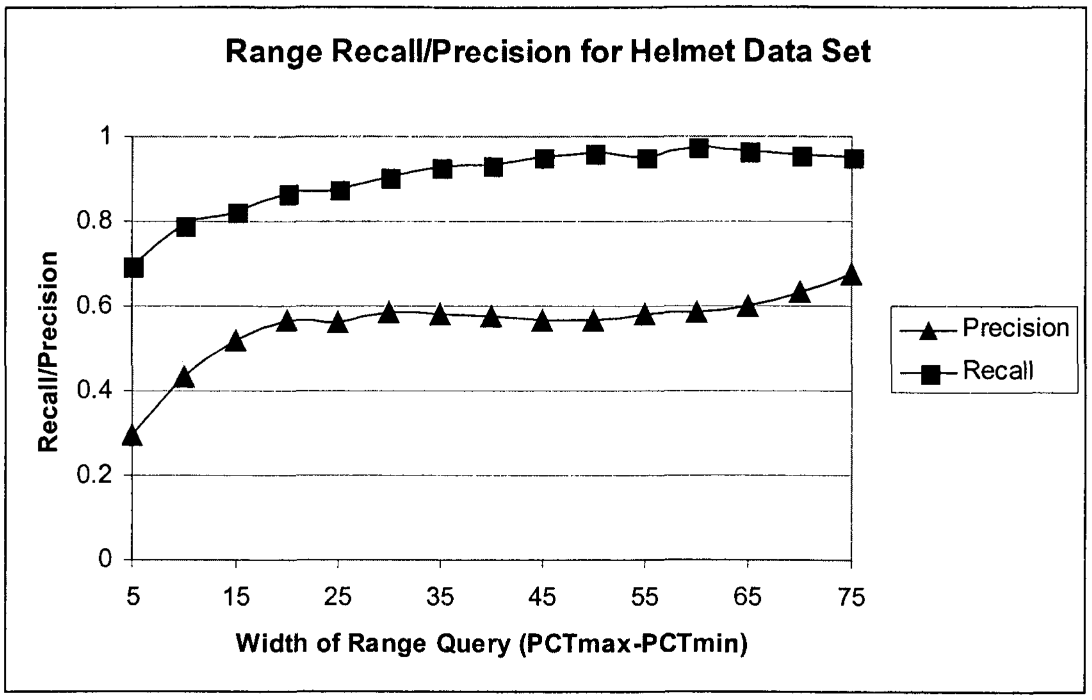


Figure 8-17a. Precision and Recall vs. Width of Range Query (Helmets)

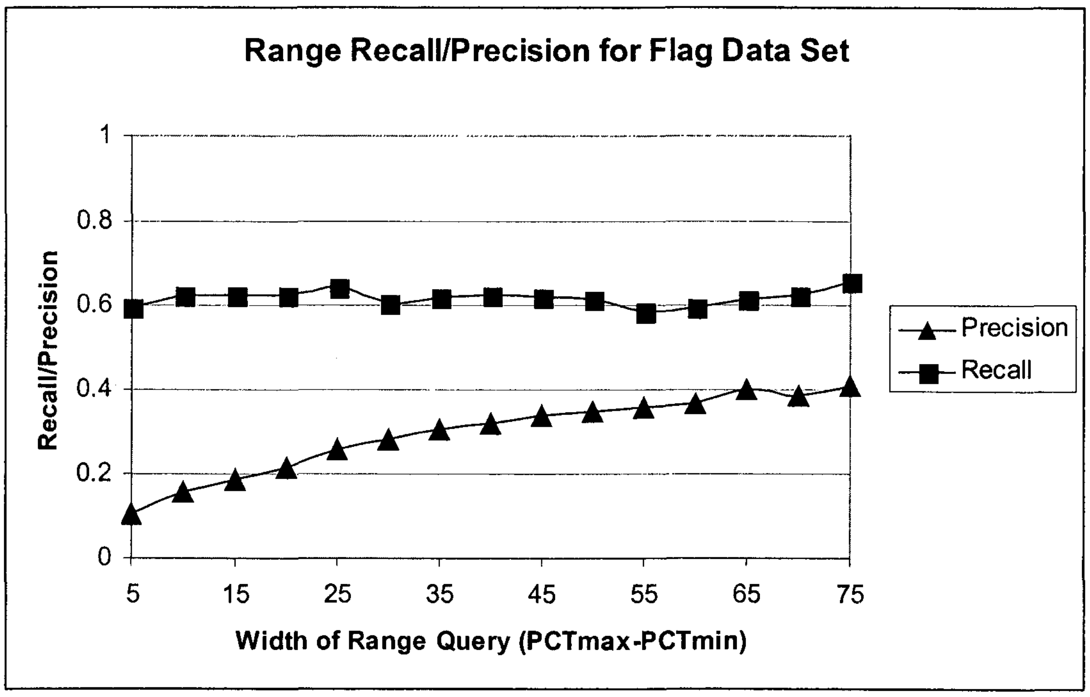


Figure 8-17b. Precision and Recall vs. Width of Range Query (Flags)

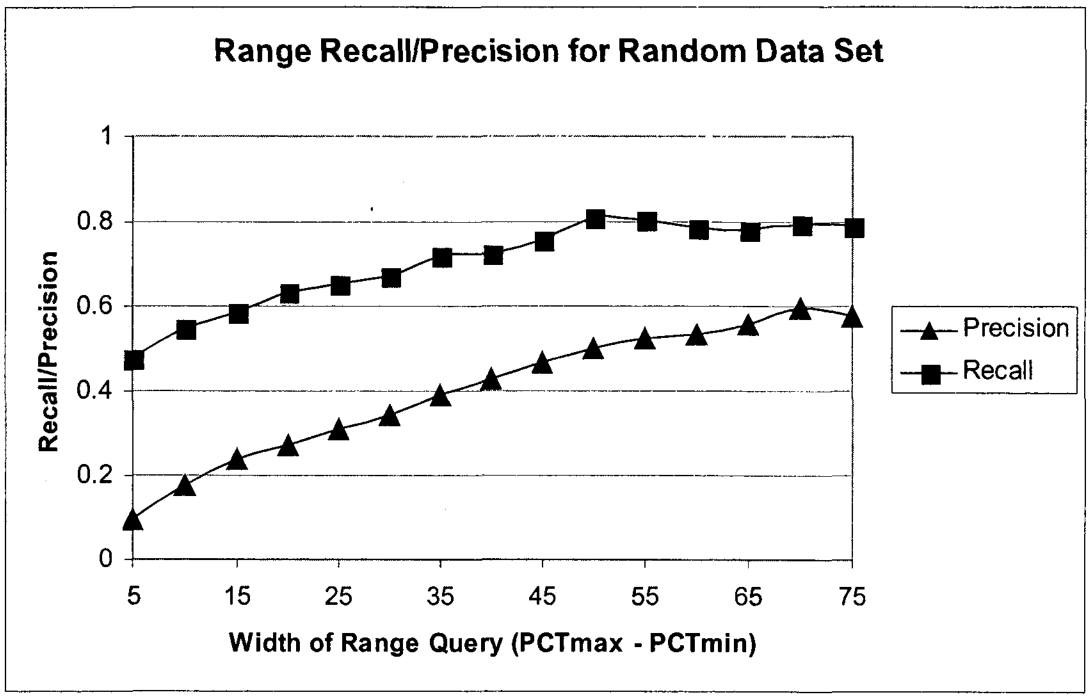


Figure 8-17c. Precision and Recall vs. Width of Range Query (Random)

The results of the above test are displayed in Figures 8-17a, 8-17b, and 8-17c for the helmet, flag, and random data sets respectively. The results display the precision and recall for the rule-based approach since it is the only approach that is not histogram-based. Since the recall is much higher than the precision in each of the tests, the results indicate that the rule-based algorithm retrieves most of the images that should be returned as a result of the query along with many more images that should not be retrieved.

The reason for the above result can be determined from the behavior of the rules. After applying the rules to the editing operations in a virtual image, the rule-based algorithm produces an interval [$\text{BOUND}_{\text{min}}$, $\text{BOUND}_{\text{max}}$]. This interval should enclose the actual percentage of pixels in the virtual image that are of the query color. Since the actual percentage of pixels is not known, however, the algorithm retrieves the image if [PCT_{min}, PCT_{max}] intersects this interval. Thus, it is possible that the intervals intersect, but the actual percentage of pixels that are of the query color is not within [PCT_{min}, PCT_{max}]. Since the actual percentage is not within this interval, the image is not relevant to the given query. Because the intervals intersect, however, the rule-based algorithm would retrieve the image. Thus, the algorithm would retrieve an image that is not relevant, which decreases its precision.

Another conclusion that is implied by the above results is that the rule-based algorithm tends to have higher rates of precision and recall as the query interval grows. Since it is easier for the bounds computed by the rules to intersect the query range if it is large, it is expected that the rule-based algorithm will retrieve more images with larger query ranges. Since the algorithm retrieves more images with larger query ranges, it is expected that the recall of the algorithm should improve and the precision of the algorithm should decline as the query range grows. The results also show that the precision is low when the query range is small, however. The reason why is that there are fewer images that can be matched by the rule-based algorithm.

The above results imply that the rule-based range query processing algorithm is more effective when used with queries with larger ranges. Thus, the rule-based algorithm is more appropriate when users search for images with a general amount of color than when they search using the precise amount of color. For example, the rule-based algorithm is more appropriate when a user can pose a query requesting the images of flags that are at least 50% red instead of requesting images that are exactly 75% red.

8.4. Performance Evaluation Summary

The performance evaluation conducted in the previous section measured the differences on the metrics of space, time, and accuracy when using the various approaches for processing queries in retrieval systems that store images virtually. This section summarizes the advantages and disadvantages of each approach.

The Binary Storage with-Histograms (BSH) algorithms have the advantage of being simple. All images can be stored in the same format so the underlying database management system only needs to manipulate one type of image. The disadvantage of the algorithms is that they consume the most space of the four approaches since no images are stored virtually. The BSH algorithms also extract histograms from all images during insertion, so this approach is also slower than the rule-based approach when inserting images.

Considering the above discussion, the BSH algorithms are best suited for applications that are most dependent on retrieval query processing time and not space or insertion time. Thus, the algorithms are the most appropriate for applications with static data sets that are searched frequently, such as medical diagnostic applications where an image taken from a patient may be compared with images of known diseases. The BSH algorithms are also appropriate for data sets that do not contain sets of images that are similar, since they do not store images virtually.

The virtual algorithms have various advantages and disadvantages. One of the advantages of the Virtual Storage with Instantiation while Searching (VSIS) algorithms is that they save space by storing images virtually. In addition, the algorithms do not store or extract any information from virtual images when they are first inserted, so the VSIS algorithms are as fast as the rule-based algorithms when inserting images. Another advantage of the VSIS algorithms is that they are histogram-based, so they produce the same results as the conventional BSH algorithms.

The main disadvantage of the VSIS algorithms is that they are extremely slow when processing retrieval queries since they instantiate all of the virtual image and extract their histograms. This means that these algorithms are not appropriate for applications that allow users to perform content-based searches frequently. Thus, the VSIS algorithms are best suited for applications in which users may constantly create and add new images to the database but rarely retrieve them. Thus, the algorithms are more useful for applications that simply archive data.

The Virtual Storage with Instantiation while Inserting (VSII) algorithms extract histograms from all images like the BSH algorithms, so they share some of the same advantages. Specifically, the BSH algorithms process the retrieval queries as quickly as the BSH algorithms, and those queries produce the same results. In addition, the VSII algorithms store images virtually, so they use less space than the BSH algorithms.

The main disadvantage of the VSII algorithms is that they are extremely slow when inserting virtual images into the underlying database management system since they instantiate the images before extracting their features. Thus, these algorithms are not appropriate for applications in which users will frequently update the database by adding new images. Alternatively, the VSII algorithms are most appropriate for many of the same applications as the BSH algorithms, which are applications with static data sets that are frequently searched. The difference is that the VSII algorithms should be used with those applications that store edited images, such as one that displays standard automobile models that differ only in color.

The rule-based algorithms implemented in the VRS prototype have the advantage that they require the significantly less space than the BSH algorithms, just as the VSIS and VSII algorithms. The main contribution of the rule-based algorithms, however, is that they do not instantiate images during the processing of insertion or retrieval queries. The result is that they are significantly faster than the VSIS algorithms when retrieving images and significantly faster than the VSII algorithms when inserting images. A potential disadvantage of the rule-based algorithms, however, is that they do not produce the same results as the BSH algorithms when processing retrieval queries. As indicated in the previous section, the algorithms are better suited for queries when users want to retrieve many images.

Considering the above information, the rule-based algorithms are best suited for applications where users insert and retrieve images frequently, and need to reduce the amount of space used by those images. Also, users should pose less-specific queries, or queries that request many images to be retrieved. Applications with these characteristics include databases for web-based stores where users want to browse many images of products that are updated quickly.

Tables 8-4a, 8-4b, and 8-4c summarize the comparison of the four approaches to retrieving images. The first column specifies the approach used and the remaining columns specify the metrics used for comparison. Each cell in the table describes the performances of the prototypes during the tests with the maximum percentage of images stored virtually, which are 29% for the helmet data set, 43% for the flag data set, and

50% for the random data set. Table 8-4a is for the helmet data set, 8-4b is for the flag data set, and 8-4c is for the random data set.

Approach	**Storage Space (MB)**	**Insertion Time (sec)**	**Nearest Neighbor Time (sec)**	**Nearest Neighbor Accuracy**
Rules (VRS)	2.97	239.65	1.34	46%
BSH	4.12	337.91	1.00	53%
VSIS	2.96	239.65	554.74	53%
VSII	2.99	891.65	1.00	53%

Table 8-4a. Comparison of Alternative Approaches (Helmet)

Approach	**Storage Space (MB)**	**Insertion Time (sec)**	**Nearest Neighbor Time (sec)**	**Nearest Neighbor Accuracy**
Rules (VRS)	3.08	817.72	3.09	34%
BSH	5.26	1414.21	2.33	45%
VSIS	3.07	817.72	5727.79	45%
VSII	3.15	7139.65	2.33	45%

Table 8-4b. Comparison of Alternative Approaches (Flag)

Approach	**Storage Space (MB)**	**Insertion Time (sec)**	**Nearest Neighbor Time (sec)**	**Nearest Neighbor Accuracy**
Rules (VRS)	0.09	6.54	2.33	73%
BSH	14.36	876.9	2.29	83%
VSIS	0.09	6.54	12174.96	83%
VSII	0.15	13049.23	2.29	83%

Table 8-4c. Comparison of Alternative Approaches (Random)

CHAPTER 9

CONCLUSION

9.1. Summary and Conclusions

MultiMedia DataBase Management Systems (MMDBMSs) focus on the storage and retrieval of images and other types of multimedia data. One requirement of these systems is to provide users a method of performing content-based searching of the multimedia data objects. Common techniques used by MMDBMSs to satisfy this requirement, such as feature extraction, usually assume that the data objects are stored in a conventional binary format.

In data sets that contain edited images, storing the edited images virtually allows them to be stored using a smaller amount of space when compared to storing them in conventional binary formats. The goal of this dissertation was to develop and evaluate multiple algorithms for performing content-based image retrieval of virtual images. In the following sections, the main points of the dissertation are summarized which includes the results of the performance evaluations.

9.1.1. Algorithms for Processing Range Queries

The first contribution of this research is the development of algorithms for processing range queries of the type "*Retrieve all images that are between* PCT_{min} *and* PCT_{max} *percent of color* C_Q" for a multimedia database management system that uses virtual images, where PCT_{min} and PCT_{max} represent percentages and C_Q represents a color in the RGB model. Three different approaches are presented for processing the above query type. The first approach uses the semantic information in virtual images to process the query. The benefit of this approach is that it is able to process queries of the above type without having to instantiate any of the virtual images by utilizing their descriptions. The algorithm has two major steps. The first step identifies the images stored in conventional binary formats that satisfy the user's query using conventional histograms. The second step in the algorithm identifies the virtual images that satisfy the query by computing the maximum and minimum bounds on the percentage of pixels that may be of color C_Q for each virtual image when it is instantiated. The bounds are computed using a series of rules, which are presented in Chapter 3. Once the bounds for a virtual image have been computed, they can then be compared to the range formed by the query arguments PCT_{min} and PCT_{max} to determine if the virtual image satisfies the query.

The second algorithm for processing range queries is the Virtual/Instantiation-Searching (VIS) approach. This approach utilizes conventional histogram techniques to retrieve virtual images by converting the images to a binary format during searching.

The final algorithm for processing range queries is the Virtual/Instantiation-Insertion (VII) approach. This approach also utilizes conventional histogram techniques to retrieve virtual images. The difference between this approach and the VIS approach is that instantiation is performed when the virtual images are inserted into the database. After each instantiation, the color histogram is extracted from the instantiated image and stored in the database.

9.1.2. Algorithms for Processing Nearest Neighbor Queries

The second contribution of this dissertation is the development of algorithms for processing nearest neighbor queries of the type "*Retrieve the k images that most resemble Q based on color*" for an MMDBMS that uses virtual images, where k represents a number and Q represents a query image. These features are assumed to be extracted from each of the binary images as they are inserted into the MMDBMS.

This dissertation presented three different approaches to processing the query, as for the processing of range queries. The rule-based approach utilizes the rules presented earlier in order to determine the colors that are contained within the virtual image without instantiating it. The VIS approach again instantiates the virtual images during retrieval in order to process the nearest neighbor queries using the conventional histogram techniques. Finally, the VII approach instantiates the virtual images during insertion and stores histograms extracted from each image in order to utilize the conventional histogram techniques later during retrieval.

9.1.3. Data Structure for Speeding up Query Processing

The third contribution of this dissertation is the development of a data structure that can be used to speed up the rule-based range query processing algorithm presented in Chapter 3. The data structure is able to save time by avoiding the processing of some of the descriptions of the virtual images. It accomplishes this by utilizing properties of some of the proposed rules for computing the minimum and maximum bounds on the percentage of pixels in a virtual image that are of a given color. Specifically, many of the rules are bound-widening meaning that they do not increase the minimum bound, and they do not decrease the maximum bound. Since the base image of a virtual image is used to initialize the minimum and maximum bounds, a virtual image whose operations correspond to bound-widening rules will satisfy a given query if its base also satisfies the query.

The proposed data structure contains two components. The first component contains a list of the binary images in the database. Each binary image is associated with the list of virtual images that all have the binary image as their base images and all contain only operations that correspond to bound-widening rules. The second component contains those virtual images that have at least one operation that does not correspond to a bound-widening rule.

9.1.4. Performance Evaluation

In Chapter 8, this dissertation provides the results of a performance evaluation comparing the conventional Binary-Histogram (BH) approach for processing retrieval queries to the rule-based, VII, and VIS approaches for processing retrieval queries with virtual images. The performance evaluation demonstrates that the BH approach uses the most space since it does not store edited images virtually. In addition, the average times used to insert an image using the VIS and rule-based approaches are expected to be shorter than the other approaches since they do not perform any processing on virtual images when they are inserted. Alternatively, the VII approach takes a long time to insert a virtual image since it instantiates the image to produce a binary version and then extracts a color histogram from the image to store into the database.

The performance evaluation also demonstrated that the rule-based approach requires much less time than the VIS approach for processing retrieval queries since the proposed approach avoids instantiating virtual images. Thus, the rule-based algorithms can process both insertion and retrieval queries quickly unlike the VIS and VII approaches while maintaining the space savings over the BH approach by storing edited images virtually.

The final result of the performance evaluation demonstrates that the rule-based approaches do not always produce the same results as the other approaches when processing retrieval queries. This is the trade-off when utilizing the rule-based approach instead of the other approaches to processing virtual images.

9.2. Directions for Future Research

Several areas of research are related to the work presented in this dissertation. One such area is to extend the work by developing techniques for retrieving images using other features besides color, such as retrieval by texture and retrieval by object shape. The reason is that although there are many applications that may benefit from virtual images, most of the real-life applications require searching based on texture and shape in addition to color. The development of these techniques is dependent on defining new rules for determining how editing operations affect texture and shape.

Another open research area concerns optimization of editing operations. When a virtual image is created inefficiently, its description may be very large. Many consequences may occur when a data set contains several virtual images with large descriptions. One such consequence is that virtual images with large descriptions require more space, which means that the space savings gained by using virtual images decreases. Another consequence is that the bounds computed by the algorithms are adjusted for each editing operation in the description of the virtual images. Thus, having a data set with large numbers of operations in the descriptions may degrade the retrieval accuracy of the algorithms. Finally, one method of instantiating a virtual image is to access the base image and apply the sequence of editing operations in the description of the virtual image on the base. If the descriptions are large, then it will take longer to apply the sequence of editing operations, which means that instantiation will be slower. Consequently, displaying the retrieved virtual images will also be slower.

Given the above problems, an MMDBMS that uses virtual images must avoid storing ones with unnecessarily long descriptions. This means that when users create virtual images inefficiently, the MMDBMS needs to be able to optimize the sequence of editing operations [Grue, 1996]. Further research is needed to develop methods of automatic optimization.

Another open research area is to evaluate the effect of using algorithms that are hybrids of the rule-based and histogram-based algorithms presented in this dissertation in order to obtain the advantages of both. For example, one such hybrid approach could store histograms corresponding to some of the virtual images in order to optimize the permanent storage space and query processing time. An open issue, then, in creating such a hybrid approach is how to determine which histograms to store that optimize those metrics.

REFERENCES

[Aars, 1999] Aars, Michael, "*Automatic Feature Extraction Using Specifications of Images*", Master's Thesis, Baylor University, 1999.

[Anna, 2000] Annamalai, Melliyal, et al., "Indexing Images in Oracle8i", *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, May 2000, pp. 539-547.

[Asla, 1999] Aslandogan, Y. Alp and Clement T. Yu, "Techniques and Systems for Image and Video Retrieval", *IEEE Transactions on Knowledge and Data Engineering*, Volume 11, Number 1, January/February 1999, pp. 56-63.

[Bach, 1993] Bach, Jeffrey R., Santanu Paul, and Ramesh Jain, "A Visual Information System for the Interactive Retrieval of Faces", *IEEE Transactions on Knowledge and Data Engineering*, Volume 5, Number 4, August 1993, pp. 619-628.

[Beck, 1990] Beckmann, Norbert, et al., "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, May 1990, pp. 322-331.

[Berc, 1996] Berchtold, Stefan, Daniel A. Keim, and Hans-Peter Kriegel, "The X-Tree: An Index Structure for High-Dimensional Data", *Proceedings of the 22nd International Conference on Very Large Databases*, 1996, pp. 28 - 39.

[Blan, 1997] Blanken, Hans, "Introduction", *Multimedia Databases in Perspective*, Chapter 1, P. M. G Apers, H. M. Blanken, and M. A. W. Houtsma (Eds.), Springer, 1997, pp. 3-11.

[Boue, 1999] Bouet, Marinette, Ali Khenchaf, and Henri Brand, "Shape Representation for Image Retrieval", *Proceedings of the 7th ACM International Conference on Multimedia*, 1999, pp. 1-4.

[Bozk, 1997] Bozkaya, Tolga and Meral Ozsoyoglu, "Distance-Based Indexing for High-Dimensional Metric Spaces", *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, May 1997, pp. 357-368.

[Bozk, 1999] Bozkaya, Tolga and Meral Ozsoyoglu, "Indexing Large Metric Spaces for Similarity Search Queries", *ACM Transactions on Database Systems*, Volume 24, Number 3, September 1999, pp. 361-404.

[Brin, 1995] Brin, Sergey, "Near Neighbor Search in Large Metric Spaces", *Proceedings of the 21st International Conference on Very Large Databases*, 1995, pp. 574-584.

[Brow, 1997] Brown, Leonard, Le Gruenwald, and Greg Speegle, "Testing a Set of Image Processing Operations for Completeness", *Proceedings of the 2nd Conference on Multimedia Information Systems*, April 1997, pp. 127-134.

[Brow, 1998] Brown, Leonard and Le Gruenwald, "Determining a Minimal and Independent Set of Image Processing Operations for a Multimedia Database System", *Proceedings of the 1998 Energy Technology Conference and Exhibition*, February 1998, pp. 1-6.

[Brow, 1998a] Brown, Leonard and Le Gruenwald, "Tree-Based Indexes for Image Data", *Journal of Visual Communication and Image Representation*, Volume 9, Number 4, 1998, pp. 300-313.

[Brow, 2001] Brown, Leonard and Le Gruenwald, "A Prototype Content-Based Retrieval System that Uses Virtual Images to Save Space", *Proceedings of the 27th International Conference on Very Large DataBases (VLDB)*, 2001, pp.693-694.

[Cheu, 1998] Cheung, King Lum and Ada Wai-chee Fu, "Enhanced Nearest Neighbour Search on the R-tree", *ACM SIGMOD Record*, Volume 27, Number 3, September 1998, pp.16-21.

[Ciac, 1997] Ciaccia, Paolo, Marco Patella, and Pavel Zezula, "M-Tree: An Efficient Access Method for Similarity Search in Metric Spaces", *Proceedings of the 23rd International Conference on Very Large Databases*, 1997, pp. 426-435.

[Come, 1979] Comer, Douglas, "The Ubiquitous B-Tree", *ACM Computing Surveys*, Volume 11, Number 2, June 1979, pp. 121-137.

[Dao, 1996] Dao, Son, Qi Yang, and Asha Vellaikal, "MB$^+$-Tree: An Index Structure for Content-Based Retrieval", *Multimedia Database Systems*, Chapter 11, Kingsley C. Nwosu, Bhavani Thuraisingham, and P. Bruce Berra (Eds.), Kluwer Academic Publishers, Boston, 1996, pp.298-317.

[Djer, 1997] Djeraba, Charbane et al., "Retrieval and Extraction by Content of Images in an Object Oriented Database", *Proceedings of the 2nd Conference on Multimedia Information Systems*, April 1997, pp. 50-57.

[Eaki, 1996] Eakins, John P., Kevin Sheilds, and Jago Boardman, "ARTISAN – A Shape Retrieval System Based on Boundary Family Indexing", *SPIE Volume 2670 Storage and Retrieval for Image and Video Databases IV*, I. K. Sethi and R. C. Jain (Eds.), SPIE Press, Bellingham, Washington, 1996, pp. 17-28.

[Eaki, 1998] Eakins, John P., Jago Boardman, and Margaret E. Graham, "Similarity Retrieval of Trademark Images", *IEEE Multimedia*, Volume 5, Number 2, April-June 1998, pp. 53-63.

[Fagi, 1998] Fagin, Ronald, "Fuzzy queries in Multimedia Database Systems", *Proceedings of the 17th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, June 1998, pp. 1-10.

[Falo, 1994] Faloutsus, C. et al., "Efficient and Effective Querying by Image Content", *Journal of Intelligent Information Systems*, Volume 3, 1994, pp. 231-262.

[Falo, 1996] Faloutsus, Christos, *Searching Multimedia Databases by Content*, Kluwer Academic Publishers, Boston, 1996.

[Flag, 2003] images from *http://www.flags.net*, accessed on January 7, 2003.

[Flic, 1995] Flickner, Myron. et al., "Query by Image and Video Content: The QBIC System", *IEEE Computer*, Volume 28, Number 9, September 1995, pp. 23-31.

[Free, 1995] Freeston, Michael, "A General Solution of the n-dimensional B-tree Problem", *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, 1995, pp. 80-91.

[Gaed, 1998] Gaede, Volker and Oliver Günther, "Multidimensional Access Methods", *ACM Computing Surveys*, Volume 30, Number 2, June 1998, pp. 170-231.

[Gong, 1994] Gong, Yihong et al, "An Image Database System with Content Capturing and Fast Image Indexing Abilities", *Proceedings of the International Conference on Multimedia Computing and Systems*, IEEE Computer, May 1994, Volume 27, Number 5, pp. 121-130.

[Gonz, 1993] Gonzales, Rafael C. and Richard E. Woods, *Digital Image Processing*, Addison-Wesley Publishing Company, Reading, MA, 1993.

[Gray, 1995] Gray, Robert S., "*Content-Based Image Retrieval: Color and Edges*", Technical Report, Dartmouth University, Identification Number PCS-TR95-252, March 1995, available at URL: http://attcomm.dartmouth.edu/~rgray/#papers.

[Gree, 1995] Greenberg, Adele Droblas and Seth Greenberg, *Fundamental Photoshop*, McGraw-Hill, Inc., Berkeley, 1995.

[Gros, 1997] Grosky, William, "Managing Multimedia Information in Database Systems", *Communications of the ACM*, Volume 40, Number 12, December 1997, pp. 73-80.

[Grue, 1996] Gruenwald, Le and Greg Speegle, "Research Issues in View-Based Multimedia Database Systems", *Proceedings of the 2nd World Conference on Integrated Design and Process Technology*, December 1996, pp. 331-336.

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

[Hafn, 1995] Hafner, James et al., "Efficient Color Histogram Indexing for Quadratic Form Distance Functions", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 17, Number 7, July 1995, pp. 729-736.

[Hear, 1997] Hearn, Donald and M. Pauline Baker, *Computer Graphics C Version*, Prentice Hall, Upper Saddle River, N.J., 1997.

[Helm, 2003] Images from http://inside99.net/Helmet_Project/index.htm, accessed on January 7, 2003.

[Hu, 1999] Hu, Shaowen, "*Instantiation of the Logical Model Language*", Baylor University, Master's Thesis, 1999.

[Jaga, 1990] Jagadish, H. V., "Spatial Search with Polyhedra", *Proceedings of the 6th International Conference on Data Engineering*, 1990, pp. 311-319.

[Jaga, 1997] Jagadish, H. V., "Content-Based Indexing and Retrieval", *The Handbook of Multimedia Information Management*, Chapter 3, William I. Grosky, Ramesh Jain, and Rajiv Mehrotra (Eds.), Prentice Hall, 1997, pp.69-93.

[Kata, 1997] Katayama, Norio, and Shin'ichi Satoh, "The SR-Tree: An Index Structure for High-Dimensional Nearest Neighbor Queries", *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, May 1997, pp. 369-380.

[Kell, 1995] Kelly, Patrick M., Michael Cannon, and Donald R. Hush, "Query by Image Example: The CANDID Approach", *SPIE Volume 2420 Storage and Retrieval for Image and Video Database III*, 1995, pp. 238-248.

[Klas, 1997] Klas, Wolfgang and Karl Aberer, "Multimedia and its Impact on Database System Architectures", *Multimedia Databases in Perspective*, Chapter 3, P. M. G. Apers, H. M. Blanken, and M. A. W. Houtsma (Eds.), Springer, New York, 1997, pp. 31-62.

[Kort, 1991] Korth, Henry F. and Abraham Silberschatz, *Database System Concepts*, McGraw-Hill, Inc., New York, 1991.

[Kuma, 1994] Kumar, Akhil, "G-Tree: A New Data Structure for Organizing Multidimensional Data", *IEEE Transactions on Knowledge and Data Engineering*, Volume 6, Number 2, April 1996, pp. 341 - 347.

[Lin, 1994] Lin, King-Ip, H. V. Jagadish, and Christos Faloutsos, "The TV-Tree: An Index Structure for High-Dimensional Data", *VLDB Journal*, Volume 3, 1994, pp 517-542.

[Lin, 2001] Lin, Shu, et al., "An Extendible Hash for Multi-Precision Similarity Querying of Image Databases", *Proceedings of the 27th International Conference on Very Large Databases*, 2001, pp. 221-230.

[Lome, 1990] Lomet, David B. and Betty Salzberg, "The hB-Tree: A Multiattribute Indexing Method with Good Guaranteed Performance", *ACM Transactions on Database Systems*, Volume 15, Number 4, December 1990, pp. 625-658.

[Mehr, 1995] Mehrotra, Rajiv and James E. Gary, "Similar Shape Retrieval in Shape Data Management", *IEEE Computer*, Volume 28, Number 9, September 1995, pp. 57-62.

[Mend, 1992] Mendenhall, William, and Terry Sincich, *Statistics for Engineering and the Sciences*, Dellen Publishing Company, San Francisco, 1992.

[NASA, 2003] Images from *http://nix.nasa.gov/browser.html*, accessed on January 7, 2003.

[Oria, 2000] Oria, Vincent et al., "DISIMA: A Distributed and Interoperable Image Database System", Demonstration, *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, May 2000, p. 600.

[Oria, 2001] Oria, Vincent et al., "Similarity Queries in the DISIMA Image DBMS", *Proceedings of the 9th ACM International Conference on Multimedia*, October 2001, pp. 475-478.

[Orte, 1997] Ortega, Michael et al., "Supporting Similarity Queries in MARS", *Proceedings of the 5th ACM International Conference on Multimedia*, 1997, pp. 403-413.

[Orte, 1998] Ortega, Michael et al., "Supporting Ranked Boolean Similarity Queries in MARS", *IEEE Transactions on Knowledge and Data Engineering*, Volume 10, Number 6, November/December 1998, pp. 905-925.

[Park, 1997] Park, Youngchoon and Forouzan Golshani, "ImageRoadMap: A New Content-Based Image Retrieval System", *Proceedings of the 8th International Conference on Database and Expert System Applications*, September 1997, Lecture Notes in Computer Science, Volume 1308, Springer, pp. 225-239.

[Park, 1999] Park, Du-sik et al., "Image Indexing using Weighted Color Histogram", *Proceedings of the 10th International Conference on Image Analysis and Processing*, 1999, pp.909-914.

[Pass, 1996] Pass, Greg, Ramin Zabih, and Justin Miller, "Comparing Images Using Color Coherence Vectors", *Proceedings of the 4th ACM International Conference on Multimedia*, 1996, pp. 65-73.

[Pbmp, 2003] http://www.acme.com/software/pbmplus/, accessed January 7, 2003.

[Rand, 2003] Images from *http://nix.nasa.gov/browser.html*, *http://www.toyota.com*, *http://c2.com/~ward/plates/*, *http://www.cs.ou.edu/~database/members.htm*, and *http://www.weathergallery.com/tornado-gallery.shtml*, accessed January 7, 2003.

[Ritt, 1996] Ritter, Gerhard X. and Joseph N. Wilson, *Handbook of Computer Vision Algorithms in Image Algebra*, CRC Press, Boca Raton, 1996.

[Robi, 1981] Robinson, John T., "The K-D-B-Tree: A Search Structure for Large Multidimensional Dynamic Indexes", *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, April 1981, pp. 10-18.

[Rous, 1995] Roussopoulos, Nick, Stephen Kelley, and Frédéric Vincent, "Nearest-Neighbor Queries", *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pp. 71-79.

[Sant, 1999] Santini, Simone and Ramesh Jain, "Similarity Measures", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 21, Number 9, September 1999, pp. 871-883.

[Scla, 1997] Sclaroff, Stan, Leonid Taycher, and Marco La Cascia, "*ImageRover: A Content-Based Image Browser for the World Wide Web*", Technical Report TR97-005, Boston University, Boston, 1997.

[Seid, 1997] Seidl, Thomas and Hans-Peter Kriegel, "Efficient User-Adaptable Similarity Search in Large Multimedia Databases", *Proceedings of the 23rd International Conference on Very Large Databases*, 1997, pp. 506-515.

[Sell, 1987] Sellis, Timos, Nick Roussopoulos, and Christos Faloutsos, "The R+-Tree: A Dynamic Index for Multidimensional Objects", *Proceedings of the 13th International Conference on Very Large Databases*, 1987, pp. 507-518.

[Shas, 1990] Shasha, Dennis and Tsong-Li Wang, "New Techniques for Best-Match Retrieval", *ACM Transactions on Information Systems*, Volume 8, Number 2, April 1990, pp. 140-158.

[Smit, 1995] Smith, John R. and Shih-Fu Chang, "*Single Color Extraction and Image Query*", IEEE Proceedings of the International Conference on Image Processing, October 1995, pp.528-531.

[Smit, 1996] Smith, John R. and Shih-Fu Chang, "VisualSEEK: A Fully Automated Content-Based Image Query System", *Proceedings of ACM Multimedia 1996*, pp. 87-98.

[Spee, 1995] Speegle, Greg, "Views of Media Objects in Multimedia Databases", *Proceedings of the International Workshop on Multimedia Database Management Systems*, August 1995, pp. 20-29.

[Spee, 1998] Speegle, Greg, Xiaojun Wang, and Le Gruenwald, "A Meta-Structure for Supporting Multimedia Editing in Object-Oriented Databases", *Proceedings of the 16th British National Conference on Databases*, July 1998, *Lecture Notes in Computer Science*, Volume 1405, Springer, pp. 89-102.

[Spee, 2000] Speegle, Greg et al., "Extending Databases to Support Image Editing", *Proceedings of the IEEE International Conference on Multimedia and Expo*, August 2000, pp.235-238.

[Steh, 2000] Stehling, Renato O., Mario A. Nascimento, and Alexandre X. Falcão, "On 'Shapes' of Colors for Content-Based Image Retrieval", *Proceedings of the 2000 ACM Workshops on Multimedia*, November 2000, pp. 171-174.

[Steh, 2002] Stehling, Renato O., Mario A. Nascimento, and Alexandre X. Falcão, "A Compact and Efficient Image Retrieval Approach Based on Border/Interior Pixel Classification", *Proceedings of the 11th International Conference on Information and Knowledge Management*, November 2002, pp. 102-109.

[Wall, 1991] Wallace, Gregory K., "The JPEG Still Picture Compression Standard", *Communications of the ACM*, Volume 34, Number 4, April 1991, pp. 30-44.

[Whit, 1996] White, David A. and Ramesh Jain, "Similarity Indexing with the SS-tree", *Proceedings of the 12th International Conference on Data Engineering*, 1996, pp. 516-523.

[Wu, 1994] Wu, Jian Kang, and Arcot Desai Narasimhalu, "Identifying Faces Using Multiple Retrievals", *IEEE Multimedia*, Volume 1, Number 3, Summer 1994, pp. 27-38.

[Yian, 1992] Yianilos, Peter N., "Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces", *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, 1992, pp. 311-321.